

Paths-based criteria and application to linear logic subsystems characterizing polynomial time

Matthieu Perrinel^a

^a*LIP, ENS Lyon, 46 alle d'Italie, 69007 Lyon, FRANCE*

Abstract

Several variants of linear logic have been proposed to characterize complexity classes in the proofs-as-programs correspondence. Light linear logic (LLL) ensures a polynomial bound on reduction time, and characterizes in this way polynomial time (*Ptime*). In this paper we study the complexity of linear logic proof-nets and propose three semantic criteria based on context semantics: stratification, dependence control and nesting. Stratification alone entails an elementary time bound, the three criteria entail together a polynomial time bound.

These criteria can be used to prove the complexity soundness of several existing variants of linear logic. We define a decidable syntactic subsystem of linear logic: *SDNLL*. We prove that the proof-nets of *SDNLL* satisfy the three criteria, which implies that *SDNLL* is sound for *Ptime*. Several previous subsystems of linear logic characterizing polynomial time (*LLL*, *mL*⁴, maximal system of *MS*) are embedded in *SDNLL*, proving its *Ptime* completeness.

1. Introduction

Motivations for a type-system capturing polynomial time. Programming is a notoriously error-prone process. The behaviours of the programs written by programmers on their first attempt often differ from their expected behaviours. Type systems can detect some of those mistakes so that programmers can correct them more easily. In this work, the property we are interested in is time complexity: the execution time of a program as a function of the size of its input. A type system *S* enforcing a polynomial bound on the time complexity of a program would be useful in several ways:

- In some real-time applications (e.g. car control systems) programs can never miss a deadline, otherwise the whole system is a failure. It is not enough to verify that the system reacted fast enough during tests, we need an absolute certainty.
- For some software, it seems enough to get an empirical estimate of the complexity by running tests. In this case, *S* could be useful to find the origin of the slowness observed during tests (this requires the type inferer to give useful information when it fails to type a term).

Email address: matthieu.perrinel@ens-lyon.fr (Matthieu Perrinel)

- In complexity theory, the main method to prove that a problem is *NP*-complete, is to define a polynomial time reduction from another *NP*-complete problem. If S is well-trusted, it could be used as a specialized proof assistant: the fact that the reduction is typable in S would increase the trust in the proof. More generally, S could be used in any proof relying on a complexity bound for a program [23, 29].

In this work, we define a subsystem $SDNLL$ of linear logic such that every proof-net normalizes in polynomial time. This property is called *Ptime soundness*. And, for every function f computable in polynomial time there exists a $SDNLL$ proof-net G_f which computes f . This property is called *Ptime extensional completeness*.

Determining if a proof-net normalizes in polynomial time is undecidable. So for every such system S , either determining if a proof-net G belongs to S is undecidable, or S is not *intensional complete*: i.e. there exist programs which normalize in polynomial time and are not typable by S . The subsystem $SDNLL$ is in the second case. We take inspiration from previous decidable type systems characterizing *Ptime* and relax conditions without losing neither soundness nor decidability. The more intensionally expressive S is (i.e. the more terms are typable by S), the more useful S is. Indeed, the three motivations for systems characterizing polynomial time we described earlier require S to type programs written by non-specialists: people who may not have a thorough understanding of S .

Linear logic and proof-nets. Linear logic (LL) [13] can be considered as a refinement of System F where we focus especially on how the duplication of formulae is managed. In linear logic, the structural rules (contraction and weakening) are only allowed for formulae of the shape $!A$:

$$\frac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B} ?C \qquad \frac{\Gamma \vdash B}{\Gamma, !A \vdash B} ?W$$

With the three following additional rules (promotion, dereliction and digging), linear logic is as expressive as System F, so the elimination of the *cut* rule (corresponding to the β -reduction of λ -calculus) is not even primitive recursive.

$$\frac{A_1, \dots, A_n \vdash B}{!A_1, \dots, !A_k \vdash !B} !P \qquad \frac{\Gamma, A \vdash B}{\Gamma, !A \vdash B} ?D \qquad \frac{\Gamma, !!A \vdash B}{\Gamma, !A \vdash B} ?N$$

However, because the structural rules are handled by 5 distinct rules, one can enforce a subtle control on the use of resources by modifying one of them. If we restrict some of those rules, it restricts the duplication of formulae. For instance, in the absence of $?D$ and $?N$ rules, the cut-elimination normalizes in elementary time [10]. The set of such proofs is defined as Elementary Linear Logic (ELL).

Proof-nets [14] are an alternative syntax for linear logic, where proofs are considered up-to meaningless commutations of rules. Proof-nets are graph-like structures where nodes correspond to logical rules. One of the reasons we use proof-nets instead of proof derivations is that context semantics, the main tool we use in this article, is much simpler to define and use in proof-nets.

Context semantics. Context semantics is a presentation of geometry of interaction [17, 11] defined by tokens traveling across proof-nets according to some rules. The paths defined by those tokens are stable by reduction so they represent the reduction of the proof-net. Context semantics has first been used to study optimal reduction [18].

Recently, it has been used to prove complexity bounds on subsystems of System T [7] and linear logic [4, 8]. In [8], Dal Lago defines for every proof-net G a weight $W_G \in \mathbb{N} \cup \{\infty\}$ based on the paths of context semantics such that, whenever G reduces to H , $W_G \geq W_H + 1$. Thus W_G is a bound on the length of the longest path of reduction starting from G . Then we can prove theorems of the shape “whenever G satisfies some property (for instance if G belongs to a subsystem such as *LLL*), W_G satisfies some bound (for instance $W_G \leq P(|G|)$ with P a polynomial and $|G|$ the size of G).”

From this point of view, context semantics has two major advantages compared to the syntactic study of reduction. First, its genericity: some common results can be proved for different variants of linear logic, which allows to factor out proofs of complexity results for these various systems. Moreover, the bounds obtained stand for any strategy of reduction. On the contrary, most bounds proved by syntactic means are only proved for a particular strategy. There are several advantages to strong bounds:

- Let us suppose we know a strong complexity bound for a system S' . We can prove the same strong complexity bound on a system S if we find an embedding ϕ of S programs in S' programs such that, whenever t reduces to u in S , $\phi(t)$ reduces to $\phi(u)$ in S' (with at least one step). We use such an embedding in Section 5.4 to prove a strong bound for λ -terms typed by *SDNLL*. If we only had a weak complexity bound for system S' , we would have to prove that the reduction from $\phi(t)$ to $\phi(u)$ matches the reduction strategy entailing the bound, which is not always possible.
- The languages we study here are confluent. However, if we consider an extension of linear logic or λ -calculus with side-effects (such as λ^R considered by Madet and Amadio in [22]), the reduction strategy influences the result of a program execution. It is important that the programmer understands the strategy. If the reduction strategy corresponded to strategies frequently used by programming languages (such as left-to-right call-by-value), it would not be a problem. However, in some cases (mL^4 for instance [3]), the strategy is rather farfetched and difficult to understand for the programmer.

Our context semantics, presented in Section 2.2, is slightly different from Dal Lago’s context semantics. In particular, Dal Lago worked in intuitionistic linear logic, and we work in classical linear logic. So the results of [8] cannot be directly applied. However most theorems of [8] have correspondents in our framework, with quite similar proofs. This is why we omit the proofs of most of the results of this section, complete proofs can be found in [26].

Our approach. Contrary to previous works, we do not directly define a linear logic subsystem. First, we define semantic criteria forbidding behaviours which can result in non-polynomial complexity. We define relations \rightarrow on boxes (special subterms of proof-nets) such that $B \rightarrow C$ means that “the number of times B is copied depends

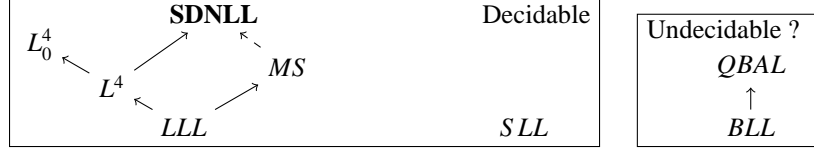


Figure 1: State of the art

on the number of times C is copied”. More precisely, we define three relations (\rightsquigarrow , \succ and \prec) representing different kinds of dependence. The acyclicity of these three relations ensures a bound on the number of times every box is copied so a bound on the length of normalization sequences.

Then (in Section 5), we define *Stratified Dependence control Nested Linear Logic* ($SDNLL$), a subsystem of linear logic such that \rightsquigarrow , \succ and \prec are acyclic on every proof-net of $SDNLL$. This entails a bound on the length of normalization for every $SDNLL$ proof-net. The relations (\rightsquigarrow , \succ and \prec) are based on the paths of context semantics. We use the syntactic restrictions to define invariants along the paths, proving that if $B \rightarrow C$ then the “types” of B and C are such that we cannot have $C \rightarrow B$. Finally, in Section 5.4, we transform $SDNLL$ into a type-system $SDNLL_\lambda$ for λ -calculus, which enforces a polynomial bound on β -reduction. This transformation is similar to the transformation of LLL [15] into $DLAL$ [5].

Previous polynomial time subsystems of Linear logic. There already exist several subsystems of linear logic characterizing polynomial time. The first such subsystem is BLL [16], which enforces *Ptime* soundness by labelling of $!$ modalities by polynomials. However, given a proof-net G , determining if G is in BLL (or its generalization $QBAL$ [9]) seems undecidable. Thus, they do not fit in our approach.

The first decidable system was LLL [15] which is defined as the proof-nets of ELL such that the contexts have at most one formula in every $!P$ rule¹. A decidable type system $DLAL$ for λ -calculus was inspired by LLL [5, 1].

Baillot and Mazza generalized ELL with a subsystem L^3 of linear logic characterizing elementary time [3]. Then they defined mL^4 and mL_0^4 , characterizing polynomial time, based on L^3 in the same way as LLL is based on ELL . In a separate direction, Roversi and Vercelli also extended LLL with MS ² [28]. Those three systems are obtained by decorating formulae with labels and adding local constraints on the labels. mL^4 , mL_0^4 and MS are trivially decidable on proof-nets: given a proof-net G there exist only a finite number of ways to label the formulae of G . One can try every possibility and check whether the labels verify the constraints. Lafont defined SLL [20], another subsystem of linear logic characterizing polynomial time. This system does not contain LLL , and none of the above generalizations of LLL contains SLL .

Figure 1 summarizes the state of the art. There is an arrow from the system S to the system T if there is a canonical embedding of S in T . The arrow between MS

¹To keep some expressivity, Girard adds a new modality \S .

²Which is a set of system rather than a unique system.

and $SDNLL$ is dotted because the embedding is only defined for one of the maximal systems of MS . In [26], we define $SwLL$ (based on the the ideas of this article) in which one can embed $SDNLL$, SLL , and every MS polynomial subsystem.

This paper extends a previous work [24] by: providing a non-trivial nesting condition, defining a syntactic subsystem based on the semantic criteria, and providing most of the proofs (in [24] the proofs are only sketched). More details, and the technical proofs omitted in this paper can be found in Perrinel's thesis [26].

2. Linear Logic and Context Semantics

2.1. Linear Logic

Linear logic (LL) [13] can be considered as a refinement of System F [12] where we focus especially on how the duplication of formulae is managed. In this work we use neither the additives (\oplus and $\&$) nor the constants. This fragment is usually named *Multiplicative Exponential Linear Logic with Quantifiers* (abbreviated by $MELL_\forall$). To simplify notations, we will abusively refer to it as *Linear Logic* (abbreviated by LL). The set \mathcal{F}_{LL} , defined as follows, designs the set of formulae of linear logic.

$$\mathcal{F}_{LL} = X \mid X^\perp \mid \mathcal{F}_{LL} \otimes \mathcal{F}_{LL} \mid \mathcal{F}_{LL} \wp \mathcal{F}_{LL} \mid \forall X. \mathcal{F}_{LL} \mid \exists X. \mathcal{F}_{LL} \mid !\mathcal{F}_{LL} \mid ?\mathcal{F}_{LL}$$

We define inductively an involution $(_)^\perp$ on \mathcal{F}_{LL} , which can be considered as a negation: $(X)^\perp = X^\perp$, $(X^\perp)^\perp = X$, $(A \otimes B)^\perp = A^\perp \wp B^\perp$, $(A \wp B)^\perp = A^\perp \otimes B^\perp$, $(\forall X.A)^\perp = \exists X.A^\perp$, $(\exists X.A)^\perp = \forall X.A^\perp$, $(!A)^\perp = ?(A^\perp)$ and $(?A)^\perp = !(A^\perp)$.

Linear logic is usually presented as a sequent calculus (as in the introduction). In this article, we will consider an alternative syntax: proof-nets [14].

Definition 1. A LL proof-net is a graph-like structure, defined inductively by the graphs of Figure 2 (G and H being LL proof-nets). Every edge e is labelled by $\beta(e) \in \mathcal{F}_{LL}$ satisfying the constraints of Figure 2. The set of edges is written \vec{E}_G .

A proof-net is a graph-like structure, whose edges are not labelled, defined inductively by the graphs of Figure 2 (G and H being proof-nets). The constraints of Figure 2 on labels are not taken into account.

For the following definitions, we supposed fixed a proof-net G .

Directed edges. The edges in the definition of proof-nets (the elements of \vec{E}_G) are directed. We will often need to consider their inverted edges: for any (l, m) (the edge from l to m), we denote its inverted edge (m, l) (the edge from m to l) by (l, m) . We define the set E_G as $\vec{E}_G \cup \{\bar{e} \mid e \in \vec{E}_G\}$. In LL proof-nets, we extend the labelling $\beta(_)$ from \vec{E}_G to E_G by $\beta(\bar{e}) = \beta(e)^\perp$.

Premises and conclusions. For any node n , the incoming edges of n (in \vec{E}_G) are named the *premises* of n . The outgoing edges of n (in \vec{E}_G) are named the *conclusions* of n . Some edges are not the premises of any node. Such edges are the *conclusions* of G .

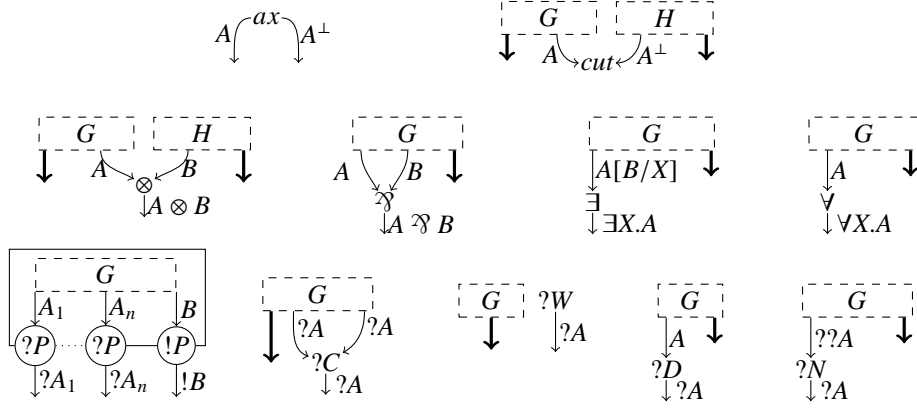


Figure 2: Construction of *LL* proof-nets. In the \forall rule, X can not be free in the other conclusions of G .

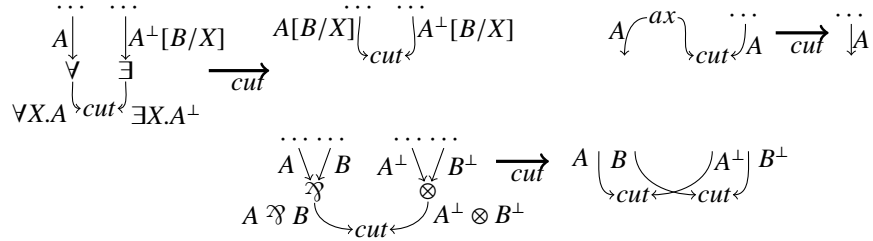


Figure 3: Non-exponential cut-elimination steps. For the \forall/\exists step, $[B/X]$ takes place on the whole net.

Boxes. The rectangle of Figure 2 with the $?P$ and $!P$ nodes is called a *box*. Formally a box is a subset of the nodes of the proof-net. We say that the edge $(m, n) \in \vec{E}_G$ belongs to box B if $n \in B$, in this case (n, m) also belongs to box B .

Let us call B the box in Figure 2. The node labelled $!P$ is the *principal door* of B , its conclusion is written $\sigma(B)$, and is named the *principal edge* of B . The $?P$ nodes are the auxiliary doors of box B . The edge going out of the i -th auxiliary door is written $\sigma_i(B)$ and is named an auxiliary edge of B . The doors of box B are considered in B , they are exactly the nodes which are in B but whose conclusions are not in B .

The number of boxes containing an element (box, node or edge) x is its *depth* written $\partial(x)$. ∂_G is the maximum depth of an edge of G . The set of boxes of G is B_G .

Cut-elimination. is a relation \rightarrow_{cut} on (*LL*) proof-nets which is related to the β -reduction of λ -calculus. Figures 3 and 4 describe the rules of cut-elimination.

Lemma 2. [14] *Proof-nets and LL proof-nets are stable under cut-elimination.*

2.2. Definition of Context Semantics

A common method to prove strong bounds on a rewriting system is to assign a weight $W_G \in \mathbb{N}$ to each term G such that, if G reduces to H , $W_G > W_H$. In *LL*, the $!P/?C$ step makes the design of such a weight hard: a whole box is duplicated, increasing the number of nodes, edges, cuts,... The idea of context semantics is to define W_G as $|A_G|$ with A_G the edges which *appear* during reduction: edges of a net G_k such that $G \rightarrow_{cut} G_1 \rightarrow_{cut} \dots \rightarrow_{cut} G_k$ ³. We can notice that whenever $G \rightarrow_{cut} H$, we have $A_H \subseteq A_G$: if $e \in A_H$ then e is an edge of a proof-net H_k with $H \rightarrow_{cut} H_1 \dots H_k$ so $G \rightarrow_{cut} H \rightarrow_{cut} H_1 \dots H_k$. Moreover, this inclusion is strict because the premises of the cut reduced between G and H are in A_G but not in A_H . Thus, $|A_G| \geq |A_H| + 2^4$.

However, such a definition of W_G would be impractical: proving a bound on W_G does not seem easier than directly proving a bound on the number of reduction steps. The solution of context semantics is to consider for every edge e of E_G , the set $Can(e)$ of *residues of e* : the elements of A_G “coming” from e . For instance, in the leftmost proof-net of Figure 5, the residues of e are $\{e, e_1, e_2, e_3, e_4\}$. The set $Can(E_G)$ of every edge residue is contained in A_G but is not always equal to it (e.g. the premises of the two cuts in the middle proof-net of Figure 5 are not residues of any edge of the leftmost proof-net so they are in A_G but not in $Can(E_G)$). Nonetheless, we still have $|Can(E_G)| \geq |Can(E_H)| + 1$ whenever $G \rightarrow_{cut} H$, so the length of any path of reduction beginning by G is at most $|Can(E_G)|$ (Theorem 11).

To bound W_G , we characterize edge residues by context semantics paths, simulating cut-elimination. Those paths are generated by *contexts* travelling across the proof-net according to some rules. The paths of context semantics in a proof-net G are exactly the paths which are preserved by cut-elimination (such paths are called *persistent* in the literature [11]). Computing those paths is somehow like reducing the proof-net. Proving bounds on the number of residues thanks to those paths rather than proving bounds directly on the reduction offers two advantages:

- Complex properties on proof-nets, which may be hard to manipulate formally, are transformed into existence (or absence) of paths of a certain shape.
- For every $G \rightarrow_{cut} H$, we have $W_G > W_H$. Thus, the length of *any* normalization sequence is bounded by W_G . The bounds obtained in this paper do not depend on the reduction strategy.

To represent lists we use the notation $[a_1; \dots; a_n]$. To represent concatenation, we use $@$: $[a_1; \dots; a_n]@[b_1; \dots; b_k]$ is defined as $[a_1; \dots; a_n; b_1; \dots; b_k]$ and $.$ represents “push” ($[a_1; \dots; a_n].b$ is defined as $[a_1; \dots; a_n; b]$). $[[a_1; \dots; a_j]]$ refers to j , the length of the list. If X is a set, $|X|$ is the number of elements of X .

A context is a pair $((e, P), T)$ composed of a *potential edge* (e, P) representing an edge residue (e is a directed edge of the proof-net) and a *trace* T used to remember some information about the beginning of the path. This information is necessary to ensure that the paths are preserved by cut-elimination. The following definitions introduce the components of potential edges and traces.

³We identify edges which are unaffected by reduction: in Figure 5, k only counts once in A_G .

⁴We can not deduce that $|A_G| > |A_H|$ because they might be both infinite.

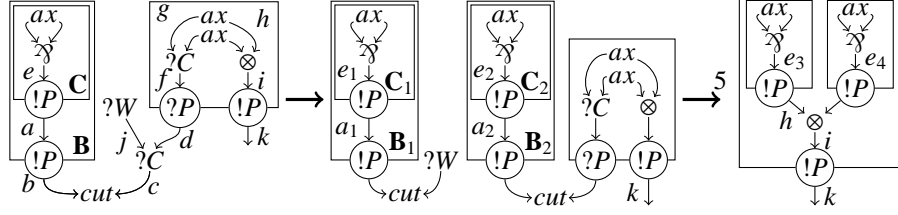


Figure 5: Cut-elimination of a proof-net.

The language *Sig* of *signatures* is defined by induction by the following grammar:

$$Sig = e \mid l(Sig) \mid r(Sig) \mid p(Sig) \mid n(Sig, Sig)$$

A signature corresponds to a list of choices of premises of $?C$ nodes, to designate a particular residue of a box. The signature $r(t)$ means: “I choose the right premise, and in the next $?C$ nodes I will use t to make my choices”. The construction $n(t, u)$ allows to encapsulate two sequels of choices into one. It corresponds to the digging rule ($!A \vdash B \rightsquigarrow !A \vdash B$, represented by the $?N$ node in proof-nets) which “encapsulates” two $!$ modalities into one. The $p(t)$ construction is a degenerated case of the n construction. Intuitively, $p(t)$ corresponds to $n(\emptyset, t)$.

A *potential* is a list of signatures: a signature corresponds to the duplication of one box, but an element is copied whenever any of the boxes containing it is cut with a $?C$ node. The set of potentials is written *Pot*. For every edge $e \in E_G$, we define $Pot(e)$ as $\{(e, P) \mid P \in Pot \text{ and } |P| = \partial(e)\}$ such pairs are named *potential edges*.

Potentials are used to represent residues. For instance, the residues of e in Figure 5, (e, e_1, e_2, e_3 and e_4) are respectively represented by the potential edges $(e, [e; e])$, $(e, [l(e); e])$, $(e, [r(e); e])$, $(e, [r(e); l(e)])$ and $(e, [r(e); r(e)])$.

A *trace element* is one of the following symbols: $\mathcal{A}_l, \mathcal{A}_r, \otimes_l, \otimes_r, \forall, \exists, !_l, ?_l$ with t a signature. A trace element means “I have crossed a node with this label, from that premise to its conclusion”. A *trace* is a non-empty list of trace elements. The set of traces is *Tra*. A trace is a memory of the path followed, up to cut-eliminations. We define duals of trace elements: $\mathcal{A}_l^\perp = \otimes_l, !_l^\perp = ?_l, \dots$ and extend the notion to traces by $[a_1; \dots; a_k]^\perp = [a_1^\perp; \dots; a_k^\perp]$.

A *context* is a tuple $((e, P), T)$ with (e, P) a potential edge and $T \in Tra$. It can be seen as a state of a token that travels around the net. It is located on edge e (more precisely its residue corresponding to P) and carries information T about its past travel. The set of contexts of G is written $Cont_G$. We extend the mapping $(\cdot)^\perp$ on contexts by $((e, P), T)^\perp = ((\bar{e}, P), T^\perp)$.

The nodes define two relations \rightsquigarrow and \hookrightarrow on contexts (Figure 6). For any rule $C \rightsquigarrow D$ presented in Figure 6, we also define the dual rule $D^\perp \rightsquigarrow C^\perp$. We define \mapsto as the union of \rightsquigarrow and \hookrightarrow . In other words, \mapsto is the smallest relation on contexts including every instance of \rightsquigarrow rules in Figure 6 together with every instance of their duals and every instance of the \hookrightarrow rule.

The rules are sound: if $((e, P), T) \mapsto ((f, Q), U)$, then $\partial(e) = |P|$ iff $\partial(f) = |Q|$. Those relations are deterministic. In particular, if $C = ((e, P), T, ?_l)$ with e the premise

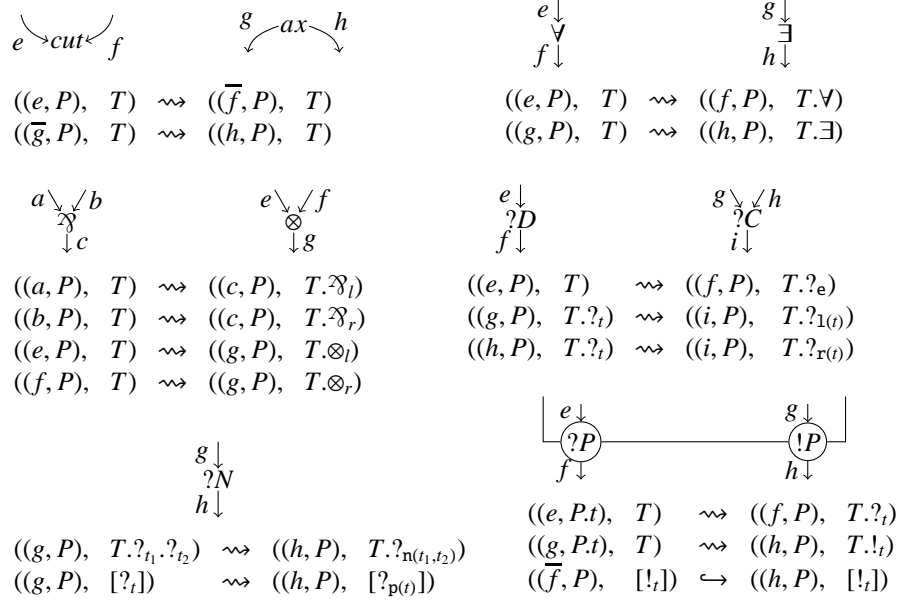


Figure 6: Rules of the context semantics

of a $?N$ node or $C = ((\overline{\sigma_i(B)}, P), T.!_t)$, the context D such that $C \mapsto D$ depends on the size of T : there is a rule in the case $T = []$ and another in the case $T \neq []$. Let us notice that \rightsquigarrow is injective (Lemma 3). It is not the case for the \mapsto relation. Indeed, if B is a box with two auxiliary doors then, for every potential P and signature t , we have $((\overline{\sigma_1(B)}, P), [!_t]) \mapsto ((\sigma(B), P), [!_t])$ and $((\overline{\sigma_2(B)}, P), [!_t]) \mapsto ((\sigma(B), P), [!_t])$.

Lemma 3. *If $C_1 \rightsquigarrow D$ and $C_2 \rightsquigarrow D$ then $C_1 = C_2$*

Finally, we can observe that for every sequence $((e_1, P_1), T_1) \rightsquigarrow ((e_2, P_2), T_2) \rightsquigarrow \dots ((e_n, P_n), T_n)$, the sequence of directed edges e_1, \dots, e_n is a path (i.e. the head of e_i is the same node as the tail of e_{i+1}). The \hookrightarrow relation breaks this property as it is non-local: it deals with two non-adjacent edges. The \mapsto -paths represent the reduction of a proof-net because they are stable along reduction. For example, the path in the first proof-net of Figure 5:

$$\begin{aligned} ((e, [r(e); l(e)], [\mathfrak{V}_r]) &\mapsto ((a, [r(e)], [\mathfrak{V}_r; !_{1(e)}]) \mapsto ((b, [], [\mathfrak{V}_r; !_{1(e)}; !_{r(e)}]) \mapsto \\ ((\overline{c}, [], [\mathfrak{V}_r; !_{1(e)}; !_{r(e)}]) &\mapsto ((\overline{d}, [], [\mathfrak{V}_r; !_{1(e)}; !_e]) \mapsto ((\overline{f}, [e], [\mathfrak{V}_r; !_{1(e)}]) \mapsto \\ ((\overline{g}, [e]), [\mathfrak{V}_r; !_e]) &\mapsto ((h, [e]), [\mathfrak{V}_r; !_e]) \mapsto ((i, [e]), [\mathfrak{V}_r; !_e; \otimes_r]) \mapsto ((k, [], [\mathfrak{V}_r; !_e; \otimes_r; !_e]) \end{aligned}$$

becomes the path $((e_3, [e; e], [\mathfrak{V}_r]) \mapsto ((h, [e]), [\mathfrak{V}_r; !_e]) \mapsto ((i, [e]), [\mathfrak{V}_r; !_e; \otimes_r]) \mapsto ((k, [], [\mathfrak{V}_r; !_e; \otimes_r; !_e])$ in the third proof-net of Figure 5.

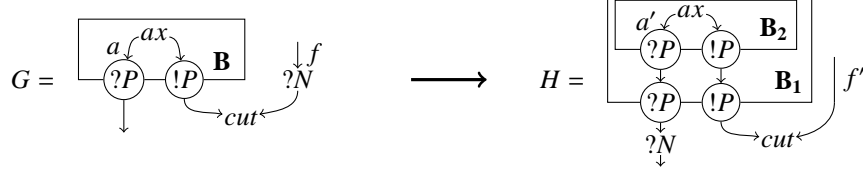


Figure 7: The potential edge $(a, [n(t_2, t_1)])$ corresponds to $(a', [t_1; t_2])$.

2.3. Dal Lago's weight theorem

As written earlier, potential edges are intended to “correspond” to residues. To precise this correspondence we first define, for every $G \rightarrow_{cut} H$ step, a partial mapping $\pi_{G \rightarrow H}(\cdot)$ from $Pot(H)$ to $Pot(G)$. For edges e which are not affected by the step, we can define $\pi_{G \rightarrow H}(e, P) = (e, P)$. If the reduction step is a $!P/?C$ step (bottom of Figure 4) and $e \in E_H$ is contained in B_l (respectively B_r) then we define $\pi_{G \rightarrow H}(e, P.t@Q) = (e, P.l(t)@Q)$ (respectively $(e, P.r(t)@Q)$) with $|P| = \partial(B)$. If the reduction step is a $!P/?N$ step, e is immediately contained in B_e and f is contained in B_i , then we define $\pi_{G \rightarrow H}(e, P.t) = (e, P.p(t))$ and $\pi_{G \rightarrow H}(f, P.t.u@Q) = (f, P.n(t, u)@Q)$. If the reduction step is a $!P/?D$ step and $e \in E_H$ belongs to G' then we define $\pi_{G \rightarrow H}(e, P@Q) = (e, P.e@Q)$. We do not detail every case and exception because the only purpose of this definition in this paper is to guide intuition. A more precise definition of the mapping is given (on contexts) in Definition 12 of [26].

Let us suppose $G_1 \rightarrow_{cut} G_2 \cdots \rightarrow_{cut} G_k$ and e' an edge of G_k . A potential edge $(e, P) \in Pot(E_{G_1})$ corresponds to e' if $\pi_{G_1 \rightarrow G_2} \circ \cdots \circ \pi_{G_{k-1} \rightarrow G_k}(e', [e; \cdots; e]) = (e, P)$.

Let $e \in E_G$, there are potential edges in $Pot(e)$ which do not correspond to residues of e . For instance, in Figure 5 a has three residues: a , a_1 and a_2 . The residue a_1 is obtained by choosing the left box during the duplication of box B , so it is represented by $(a, [l(e)])$. Similarly, a and a_2 are represented by $(a, [e])$ and $(a, [r(e)])$. However, $(a, [r(l(e))])$ does not represent any residue. The potential node $(a, [r(l(e))])$ means that whenever the box B_2 is cut with a $?C$ node, we choose the left box. But this situation never happens. It can be observed by the following path:

$$((\sigma(B), []), [!_{r(l(e))}]) \mapsto ((\bar{c}, []), [!_{r(l(e))}]) \mapsto ((\bar{d}, []), [!_{l(e)}]) \mapsto ((k, []), [!_{l(e)}]) \not\mapsto$$

The $l(_)$ has not been used because we did not encounter a second $?C$ node. On the contrary, the signatures corresponding to residues are entirely used:

$$\begin{aligned} ((\sigma(B), []), [!_e]) &\mapsto^0 ((b, []), [!_e]) \\ ((\sigma(B), []), [!_{l(e)}]) &\mapsto^2 ((\bar{j}, []), [!_e]) \quad ((\sigma(B), []), [!_{r(e)}]) \mapsto^2 ((\bar{d}, []), [!_e]) \end{aligned}$$

This is why, in the absence of $?N$ nodes, we define the *canonical potentials* of $e \in B_{\partial(e)} \subset \cdots B_2 \subset B_1$ as the potential edges $(e, [p_1; \cdots; p_{\partial(e)}])$ such that, for $1 \leq i \leq \partial(e)$, we have $((\sigma(B_i), [p_1; \cdots; p_{i-1}]), [!_{p_i}]) \mapsto^* ((_, _), [!_e])$ (throughout the article, we use $_$ to denote an object whose name and value are not important to us, for example $C \mapsto _$ means $\exists D \in Cont_G, C \mapsto D$).

Now we will consider what happens when $?N$ nodes are allowed. Let us consider the node a in Figure 7. The residues of a are exactly the residues of a' and a itself, and “ $(a', [t_1; t_2])$ corresponds to a residue of a ” is successively equivalent to:

$$\begin{aligned} & \left\{ \begin{array}{l} ((\sigma(B_2), [t_1]), [!_{t_2}]) \mapsto^* ((\neg, \neg), [!_e]) \\ ((\sigma(B_1), []), [!_{t_1}]) \mapsto^* ((\neg, \neg), [!_e]) \end{array} \right\} \Leftrightarrow \left\{ \begin{array}{l} ((\overline{f'}, []), [!_{t_2}; !_{t_1}]) \mapsto^* ((\neg, \neg), [!_e]) \\ ((\overline{f'}, []), [!_{t_1}]) \mapsto^* ((\neg, \neg), [!_e]) \end{array} \right\} \\ \Leftrightarrow & \left\{ \begin{array}{l} ((\overline{f}, []), [!_{t_2}; !_{t_1}]) \mapsto^* ((\neg, \neg), [!_e]) \\ ((\overline{f}, []), [!_{t_1}]) \mapsto^* ((\neg, \neg), [!_e]) \end{array} \right\} \Leftrightarrow \left\{ \begin{array}{l} ((\sigma(B), []), [!_{n(t_2, t_1)}]) \mapsto^* ((\neg, \neg), [!_e]) \\ ((\sigma(B), []), [!_{p(t_1)}]) \mapsto^* ((\neg, \neg), [!_e]) \end{array} \right\} \end{aligned}$$

Thus, $(a, [n(t_2, t_1)])$ corresponds to a residue of a iff both $n(t_2, t_1)$ and $p(t_1)$ are entirely used by their \mapsto -paths. Let us notice that a box may encounter several $?N$ nodes during cut-elimination. To check every case, we define a relation \sqsubseteq on signatures such that, in particular, $n(t_2, t_1) \sqsubseteq n(t_2, t_1)$ and $n(t_2, t_1) \sqsubseteq p(t_1)$.

Definition 4. A signature is standard if it does not contain the constructor p . A signature t is quasi-standard iff for every subtree $n(t_1, t_2)$ of t , the signature t_2 is standard.

The binary relation \sqsubseteq on Sig is defined by induction as follows: $e \sqsubseteq e$ and, if we suppose that $t \sqsubseteq t'$, then $l(t) \sqsubseteq l(t')$, $r(t) \sqsubseteq r(t')$, $p(t) \sqsubseteq p(t')$, $n(u, t) \sqsubseteq p(t')$ and $n(t, u) \sqsubseteq n(t', u)$. We write that t' is a *simplification* of t , when $t \sqsubseteq t'$. We write $t \sqsubset t'$ for “ $t \sqsubseteq t'$ and $t \neq t'$ ”. We can observe that \sqsubseteq is an order and \sqsubset a strict order.

Lemma 5 ([26]). Let $t \in Sig$, then \sqsubseteq is a total order on $\{u \in Sig \mid t \sqsubseteq u\}$.

Definition 6. A context $((e, P), [!_t]@T)$ is said quasi-standard if t is quasi-standard and every signature in P and T is standard.

If $u \sqsupseteq t$ with t standard $((\sigma(B), P), [!_t])$ is quasi-standard, and quasi-standard contexts are stable by \mapsto [26]. So every context we study in this work is quasi-standard.

We capture the notion of residue by *canonical potentials*. The definition of canonical potentials relies on *copies*. A copy represents the choices for one box, a canonical potential for an element x is a list of copies: one copy for each box containing x .

Definition 7. A copy context is a context of the shape $((e, P), [!_t]@T)$ such that for every $u \sqsupseteq t$, there exists a path of the shape $((e, P), [!_u]@T) \mapsto^* ((\neg, \neg), [!_e]@_)$.

Let $(B, P) \in Pot(B_G)$, the set $Cop(B, P)$ of copies of (B, P) is the set of standard signatures t such that $((\sigma(B), P), [!_t])$ is a copy context.

For instance, in Figure 5, the copies of $(B, [])$ are e , $l(e)$ and $r(e)$ which respectively corresponds to B itself, B_1 and B_2 . So $(C, [l(e)])$ and $(C, [r(e)])$ correspond respectively to C_1 and C_2 . We can notice that C_2 is duplicated while C_1 can not be duplicated, in terms of context semantics $((\sigma(C), [r(e)]), [!_{l(e)}]) \mapsto^5 ((\bar{g}, [e]), [!_e])$ so $l(e)$ is a copy of $(C, [r(e)])$ while $((\sigma(C), [l(e)]), [!_{l(e)}]) \mapsto^3 ((\bar{j}, []), [!_{l(e)}; !_e]) \not\mapsto$ so $l(e)$ is not a copy of $(C, [l(e)])$.

Definition 8. Let x be an edge (resp. box, node) of G with $x \in B_{\partial(x)} \subset \dots \subset B_1$. The set $Can(x)$ of canonical edges (resp. box, node) for x is the set of tuples $(x, [p_1; \dots; p_{\partial(x)}])$ with $p_1, \dots, p_{\partial(x)}$ signatures such that:

$$\forall 1 \leq i \leq \partial(x), p_i \in Cop(B_i, [p_1; \dots; p_{i-1}])$$

For instance, in the proof-net of Figure 5, we have $Cop(B, []) = \{e, l(e), r(e)\}$, $Cop(C, [e]) = Cop(C, [l(e)]) = \{e\}$ and $Cop(C, [r(e)]) = \{e, l(e), r(e)\}$. So, by definition, $Can(e) = \{e\} \times \{[e; e], [l(e); e], [r(e); e], [r(e); l(e)]; [r(e); r(e)]\}$. Those canonical potentials correspond respectively to e, e_1, e_2, e_3 and e_4 . We can notice that $|Can(e)| = 5$. In the middle proof-net of this Figure, we have $Can(e_1) = \{(e_1, [e; e])\}$ and $Can(e_2) = \{(e_2, [e; e]), (e_2, [e; l(e)]), (e_2, [e; r(e)])\}$ so $|Can(e_1)| + |Can(e_2)| = 4 < |Can(e)|$ (the number of residues decreases because there is no edge corresponding to $(e, [e, e])$ in the middle proof-net).

The set of canonical edges of G is represented by $Can(E_G)$. Let us notice that the canonical edges for e only depend on the boxes containing e : if e and f are contained in the same boxes then $Can(e) = \{(e, P) \mid (f, P) \in Can(f)\}$.

Definition 9. For any proof-net G , we define $W_G = |Can(E_G)| \in \mathbb{N} \cup \{\infty\}$.

In [26], to prove that W_G is a bound on reduction, we first build a strict injection from the canonical nodes of H to the canonical nodes of G . This injection is based on a mapping from contexts of H to contexts of G which preserves \mapsto -paths. Then we prove that the number of canonical nodes is bounded by W_G .

Definition 10. Let us suppose that $G \rightarrow_{cut} H$ then we defined (in [26]) a partial mapping $\pi(\cdot)$ from $Cont_H$ to $Cont_G$ such that, whenever $\pi(C)$ and $\pi(D)$ are defined,

$$C \mapsto^* D \Rightarrow \pi(C) \mapsto^* \pi(D) \quad C \mapsto^+ D \Leftarrow \pi(C) \mapsto^+ \pi(D)$$

Theorem 11 is a slight variation of the Lemma 6 of Dal Lago in [8]. This result allows to prove strong complexity bounds for several systems.

Theorem 11 ([26]). *If G is a normalizing proof-net, then $W_G \in \mathbb{N}$. The length of any path of reduction, and the size of any proof-net of the path, is bounded by W_G .*

Execution time depends on the implementation of proof-nets and *cut*-elimination. In a basic implementation based on graphs, every step can be done in constant time except for the box rules, which can be done in a time linear in the size of the box, so linear in the size of the proof-net. Thus, according to Theorem 11, the execution time of G is in $O(W_G^2)$. The complexity classes we study in this article are stable by polynomial. Thus, to establish the soundness of a *LL* subsystem with respect to polynomial time/elementary time, it is enough to prove a polynomial/elementary bound on W_G .

Lemma 12 ([26]). *Let G be a normalizing proof-net, there is no path of the shape $((e, P), [!_l]) \mapsto^+ ((e, P), [!_u])$ with (e, P) a canonical edge.*

3. Paths criteria for elementary time

3.1. History and motivations

A stratification refers to a restriction of a framework, which forbids the contraction (or identification) of two subterms belonging to two morally different “strata”. Stratification restrictions might be applied to several frameworks (naive set theory, linear

logic, lambda calculus and recursion theory) to entail coherence or complexity properties [3]. To define a stratification condition on Linear Logic we define, for every proof-net G , a *stratification relation* $>$ between the boxes of G . Then, we consider that B belongs to a higher stratum than C if $B(>)^+C$. The relation $>$ must be defined such that there exists a function f such that:

$$|Cop(B, P)| \leq f \left(\max_{\substack{B > C \\ (B, P) \in Pot(B)}} |Cop(C, Q)|, |E_G| \right) \quad (1)$$

One says that G is $>$ -stratified if $>$ is acyclic. In this case, for every box B of G , we define the $>$ -stratum of B_1 (written $s_>(B_1)$) as the greatest $i \in \mathbb{N} \cup \{\infty\}$ such that there exists $B_2, \dots, B_i \in S$ such that $B_1 > B_2 > \dots > B_i$. We define $|>|$ as $\max_{B \in B_G} s_>(B)$. If t is $>$ -stratified, the $>$ -stratum of every box is in \mathbb{N} because $|B_G|$ is finite. Thus, one can bound $|Cop(B)|$ by induction on $s_>(B)$ (thanks to Equation 1). Because $W_G \leq |E_G| \cdot (\max_{(B, P) \in Pot(B_G)} |Cop(B, P)|)^{|B_G|}$, this gives us a bound on W_G .

In most previous works, the stratum $s(\cdot)$ is rather explicit while $>$ is left implicit (it can be defined by “ $B > C$ iff $s(B) > s(C)$ ”). Concretely, in [15] and [10], the stratum of a box is defined as its depth (the number of boxes containing it). To enforce Equation 1, digging and dereliction ($?N$ and $?D$ nodes) are forbidden. In [3], Baillot and Mazza label the edges with a natural number. To enforce Equation 1, Baillot and Mazza define some local conditions that those labels have to satisfy. Those works are presented as subsystems of Linear Logic: *ELL* [15] and *L³* [3]. In both cases, the function f in Equation 1 is an elementary function (tower of exponential of fixed height). Because this class of functions is stable by composition and maximum, *ELL* and *L³* proof-nets normalize in a number of steps bounded by an elementary function of its size, and this function only depends on $\max_{B \in B_G} s(B) \leq |B_G|$.

When they defined *L³* [3], Baillot and Mazza did more than improving the intensional expressivity of *ELL*, they showed that “exponential boxes and stratification levels are two different things”. This clarified the notion of stratification and enabled the present work. Here, we go further in that direction: we disentangle three principles (stratification, dependence control and nesting) which are implicit in *LLL* and *L⁴*. These principles are presented as the acyclicity of relations on boxes (respectively \rightsquigarrow , \succ and \prec) whose intuitive meanings are described below. The meanings are voluntarily vague because those principles are not limited to the representations given in this paper, there are many variations possible [26]. The intuitive meanings are not given in terms of linear logic but in the larger setting of models of computation based on rewriting. Indeed, we applied those principles both to linear logic and λ -calculus. We believe them to be relevant in other frameworks based on rewriting such as interaction nets, recursion theory and term rewriting systems. In this larger setting, the relations are between *parts* of a programs (boxes for linear logic, subterms for λ -calculus).

- Stratification (Section 3.2): $B \rightsquigarrow C$ means that B will interact with a part C' (i.e. during reduction there is a rewriting step involving B and C') which will be created by a rewriting rule involving C . For instance, let us consider $t = \lambda x.(\lambda y.(y)\lambda w.w)\lambda z.(z)x$, we have $\lambda w.w \rightsquigarrow \lambda z.(z)x$ because $t \rightarrow_\beta \lambda x.(\lambda z.(z)x)\lambda w.w \rightarrow_\beta \lambda x.(\lambda w.w)x \rightarrow_\beta \lambda x.x$ so the last step is a rewriting step involving both $\lambda w.w$ and $B' = x$, which is created during a step involving $\lambda z.(z)x$ (the second step).

- **Dependence control** (Section 4.1): $B \succ C$ means that several parts of C will be substituted by B . Those parts will not be duplicated inside C . For instance, let us consider $t = \lambda y.(\lambda x.(x)(x)(\lambda w.w)y)\lambda z.z$, we have $\lambda z.z \succ (\lambda x.(x)(x)(\lambda w.w)y)$ because the two occurrences of x in $\lambda x.(x)(x)(\lambda w.w)y$ will indeed be replaced by $\lambda z.z$. None of those occurrences of x will be duplicated during a normalization of $\lambda x.(x)(x)(\lambda w.w)y$.
- **Nesting** (Section 4.2): $B \prec C$ means that a part of C will be substituted by B . Those free variables may be duplicated inside C . For instance, let us consider the λ -term $t = (\lambda y.(\lambda x.(y)x)\lambda z.z)\lambda w.(w)w$ we have $\lambda z.z \prec \lambda x.(y)x$ we can notice that the occurrence of x in $\lambda x.(y)x$ will indeed be replaced by $\lambda z.z$. This occurrence of x may be duplicated, with the reduction $t \rightarrow_\beta (\lambda x.(\lambda w.(w)w)x)\lambda z.z \rightarrow_\beta (\lambda x.(x)x)\lambda z.z$.

The acyclicity of \rightsquigarrow entails an elementary bound on W_G (Theorem 35), the acyclicity of the three relations entails a polynomial bound on W_G (Corollary 42). We want to find characterizations of complexity classes which are as intensionally expressive as possible. So we try to find the smallest possible relation $>$ (with respect to inclusion) whose acyclicity entails a bound of the shape of Equation 1. Indeed if, for every proof-net the relation R_1 on boxes is a subset of R_2 , then the acyclicity of R_2 implies the acyclicity of R_1 . So more proof-nets are R_1 -stratified than R_2 -stratified.

We want to prove a bound on the number of copies of boxes. Let us consider a potential box (B, P) and a copy t of (B, P) , by definition of copies there exists a path $((\sigma(B), P), [!_t]) \mapsto^* ((e, Q), [!_e])$. Our idea to prove Corollary 42 is to determine entirely t from a partial information on (e, Q) and on the \hookrightarrow steps of the path. Because there is a bounded number of possibilities for those information, we have a bound on the number of copies of (B, P) .

- *Stratification*: When \rightsquigarrow is acyclic, one can *trace back* \rightsquigarrow -paths: let us suppose that $C_k \rightsquigarrow^* C_1 \rightsquigarrow C_0$, with some partial information on C_0 we can deduce a partial information on C_1, C_2, \dots, C_k . In particular, we can deduce the edges of all those contexts.
- *Dependence control*: When \succ is acyclic, one can *trace back* the \hookrightarrow steps. Thus, if \rightsquigarrow and \succ are acyclic and $C_k \mapsto^* C_1 \mapsto C_0$, we only need a bounded amount of information to deduce the edges of the contexts. This gives us a bound on the number of sequences e_k, \dots, e_1, e_0 of edges such that there exists a path of the shape $((\sigma(B), P), [!_t]) \mapsto ((e_k, _), _) \mapsto \dots ((e_1, _), _) \mapsto ((e_0, _), [!_e])$.
- *Nesting*: If there is no $?N$ node, then a copy t of (B, P) is a list of 1 and r which is entirely determined by the sequence e_k, \dots, e_0 of edges of the path $((\sigma(B), P), [!_t]) \mapsto ((e_k, _), _) \mapsto \dots ((e_1, _), _) \mapsto ((e_0, _), [!_e])$. Combined with the acyclicity of \rightsquigarrow and \succ , this gives us a bound on $|Cop(B, P)|$.

3.2. Definition of \rightsquigarrow -stratification

To prove the complexity bounds for *ELL* and *LLL*, one usually uses a round-by-round cut-elimination procedure. During round i , we reduce every cut at depth i . We

can bound the number of $?C$ node residues at depth $i + 1$ and, because the boxes at depth i can only be duplicated by $?C$ nodes at depth $i + 1$, it gives us a bound on the number of times boxes at depth $i + 1$ are duplicated. We will proceed similarly: we will prove a bound on the number of nodes (in particular the $?C$ and $?N$ nodes) obtained after i rounds of cut-elimination, and prove that it gives us a bound on the number of duplication during round $i + 1$ by tracing back paths corresponding to copies from the $((e, P), [!_e])$ context⁵ back to $((\sigma(B), P), [!_t])$ and showing that the potential edge (e, P) (corresponding to a residue) determines t in a unique way.

To understand the definition of \rightsquigarrow , let us first define a relation \rightarrow on boxes by: $B \rightarrow C$ iff there exists a path of the shape $((\sigma(B), _), [!_t]) \mapsto^* ((e, _), _)$ with $e \in C$.

Let us notice that, if $((\sigma(B), P), [!_t]) \mapsto^* ((e_k, P_k), [!_{t_k}] @ T_k) \rightsquigarrow^k ((e_0, P_0), [!_{t_0}])$, one only needs to know e_0 and P_0 to deduce $(e_i, P_i, T_i)_{1 \leq i \leq k}$ (because \rightsquigarrow is injective). By definition of \rightarrow , for every box C containing e_i , we have $B \rightarrow C$. Thus, there are only $|E_G| \cdot (\max_{B \rightarrow C} |Cop(C, Q)|)^{\theta_G}$ such paths (it is enough to fix $e_0 \in E_G$ and a copy for every box containing e_0).

The idea of this section is to identify *unnecessary* $B \rightarrow C$ pairs. It is to say, boxes B and C such that $B \rightarrow C$ but tracing back \rightsquigarrow -paths originating from $((\sigma(B), P), [!_t])$ does not depend on an element of $Cop(C, Q)$. The first such example is whenever $B \subset C$ and no \mapsto path from $((\sigma(B), P), [!_t])$ to $((e, R), [!_u])$ leaves the box C . In this case, the signature corresponding to C never changes along the path. So, whenever $((\sigma(B), P), [!_t]) \mapsto^* ((e_k, P_k), [!_{t_k}]) \rightsquigarrow^k ((e_0, P_0), [!_{t_0}])$ the signature corresponding to C is the same in P, P_k and P_0 . This signature never goes to the trace, so knowing it is not necessary to trace back the path. In this case, knowing $\max_Q |Cop(C, Q)|$ is not necessary to bound the number of \rightsquigarrow -paths originating from $((\sigma(B), P), [!_t])$.

Thus, $B \rightarrow C$ couples are necessary only if there is a \mapsto path from $((\sigma(B), P), [!_t])$ which enters C by one of its doors (either auxiliary or principal). In fact, we prove that the $B \rightarrow C$ couples are necessary only if there is a \mapsto path from $((\sigma(B), P), [!_t])$ which enters C by its *principal* door. To understand why, we study an example. In Figure 8, if $((\sigma(B), P), [!_t]) \rightsquigarrow^* ((\bar{d}, q), [!_e])$, we need to know q to trace back the path (i.e. to deduce the list of edges of those paths) because:

$$\begin{cases} ((\sigma(B), [r(e)]), [!_{r(e)}]) \rightsquigarrow^4 ((g, []), [!_e; ?_{1(e)}]) \rightsquigarrow^3 ((\bar{d}, [1(r(e))]), [!_e]) \\ ((\sigma(B), [r(e)]), [!_{1(e)}]) \rightsquigarrow^4 ((h, []), [!_e; ?_{r(e)}]) \rightsquigarrow^3 ((\bar{d}, [r(r(e))]), [!_e]) \end{cases} \quad (2)$$

So $B \rightarrow C$ is a necessary pair. Tracing those paths backwards, the difference in the potential corresponding to C becomes a difference in a $?_t$ trace element (in the $((\sigma(C), []), [!_e; ?_q]) \mapsto ((\bar{d}, [q]), [!_e])$ step). And because of this difference on a $?_t$ trace element, the reverse paths separate when the paths cross a $?C$ node downwards: $((g, []), [!_e; ?_e]) \mapsto ((f, []), [!_e; ?_{1(e)}])$ and $((h, []), [!_e; ?_e]) \mapsto ((f, []), [!_e; ?_{r(e)}])$.

On the contrary, if $((\sigma(D), P), [!_t]) \rightsquigarrow^* ((\bar{w}, [q_A; q_B]), [!_e])$, we only need to know q_B to trace back the path. Indeed the paths do not enter A by its principal door, so q_A can only appear on $!_t$ trace elements, never on $?_t$ trace elements.

⁵One can observe that we can restrict e to be either the principal door of B , or a (reverse) premise of a $?C$ or $?N$ node, because crossing those nodes upwards are the only step modifying the signature of the left-most trace element.

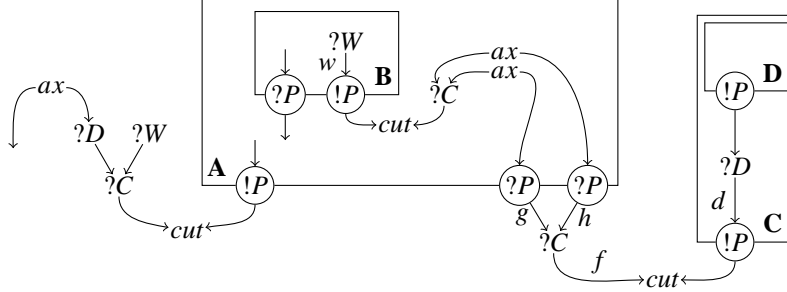


Figure 8: $D \twoheadrightarrow A$ but it is an “unnecessary” couple because $|Cop(B, P)|$ does not depend on $|Cop(A, [])|$.

We define a relation \rightsquigarrow between boxes of proof-nets. $B \rightsquigarrow C$ means that there is a path beginning by the principal door of B which enters C by its principal door.

Definition 13. Let $B, C \in B_G$, we write $B \rightsquigarrow C$ if there is a path of the shape:

$$((\sigma(B), P), [!_t]) \rightsquigarrow^* ((\overline{\sigma(C)}, Q), T)$$

We can notice that for every proof-net, $\rightsquigarrow \subseteq \twoheadrightarrow$. For example, in the proof-net of Figure 8, we have $B \twoheadrightarrow A$, $B \twoheadrightarrow C$, $D \twoheadrightarrow A$, $D \twoheadrightarrow B$ and $D \twoheadrightarrow C$ while the only pair for \rightsquigarrow are $B \rightsquigarrow C$ and $D \rightsquigarrow B$.

As shown in Equation 2, to trace back the \rightsquigarrow -path from $((\sigma(B), [r(e)]), [!_{r(e)}])$ to $((\overline{d}, [q]), [!_e])$ one needs information on $q = l(r(e))$. However, let us notice that it is not necessary to know q entirely. The only information needed to trace back the path is that it is of the form $l(x)$. Knowing that $x = r(e)$ is useless because the information in x would only be used if the path entered A by its principal door and that is not the case.

The following intuitions (formalized in Section 3.3) capture the notion of the information needed to trace back the paths. As we stated earlier, a canonical potential of a box corresponds to a residue of this box along reduction, a \mapsto_S -canonical potential of a box corresponds to a residue obtained without firing cuts involving the principal door of a box outside S . It is to say, a \mapsto_S -canonical potential of a box corresponds to a residue of this box along reduction such that, for every step of this reduction involving the principal door of a box B , B is a residue of a box of S .

More formally, we first define the \mapsto_S -copies of (B, P) as the copies t of (B, P) such that in the paths $((\sigma(B), P), [!_t]) \mapsto^* ((_, _), [!_e])$, every \hookrightarrow step of the path is on a box of S . For instance, in the proof-net of Figure 8, the $\mapsto_{\{C\}}$ -copies of $(C, [])$ are $\{e, l(e), r(e)\}$ while the $\mapsto_{\{C, A\}}$ -copies of $(C, [])$ are $\{e, l(e), r(e), l(l(e)), l(r(e)), r(l(e)), r(r(e))\}$. Then, we define \mapsto_S -canonical potentials from the notion of \mapsto_S -copies in the same way as we defined canonical potentials from the notion of copies.

Let us suppose that we know that $_ \rightsquigarrow^7 ((\overline{d}, [q]), [!_e])$ and $_ \rightsquigarrow^7 ((\overline{d}, [q']), [!_e])$ and we want to prove that those paths take the same edges. We only need to know that the $\mapsto_{\{C\}}$ -copies of $(C, [])$ “corresponding” to q and q' are equal. We define x (resp. x') as the “biggest” $\mapsto_{\{C\}}$ -copy of $(C, [])$ which is a “truncation” of q (resp. q'). For instance, if $q = r(l(e))$ and $q' = r(r(e))$, then $q \neq q'$ but we have $x = x' = r(e)$. This is enough

to know that q and q' are of the shape $r(_)$ and this information is enough to trace back the paths, so to prove that the paths take the same edges.

The \mapsto_S -copy of (B, P) corresponding to t is written $((\sigma(B), P), [!_t])^{\mapsto_S}$. It represents the part of t which is used if we refuse the \hookrightarrow steps over the potential boxes which are not in S . For instance, in Figure 8, $((\sigma(C), []), [!_{r(1(e))}])^{\mapsto_{\{C\}}} = r(e)$ because, if we refuse to jump over $(A, [])$, only $r(_)$ is consumed in the \mapsto paths starting from this context. Then, $(e, P)^{\mapsto_S}$ is defined from the $((\sigma(B), P), [!_t])^{\mapsto_S}$ construction in the same way as canonical potentials are defined from copies.

3.3. Restricted copies and canonical potentials

Now that we gave the intuitions, we can state the formal definitions.

Definition 14. Let G be a proof-net and $S \subset B_G$. We define \mapsto_S and \rightsquigarrow_S as follows:

$$C \mapsto_S D \Leftrightarrow \begin{cases} C \mapsto D \\ \text{If } C = ((\sigma(B), P), [!_t]), \text{ then } B \in S \end{cases}$$

$$C \rightsquigarrow_S D \Leftrightarrow \begin{cases} C \rightsquigarrow D \\ \text{If } D = ((\overline{\sigma(B)}, P), T.?,_t), \text{ then } B \in S \end{cases}$$

If \mapsto corresponds to cut-elimination, \mapsto_S corresponds to cut-elimination restricted by allowing reduction of cuts involving the principal door of a box B only if B is a residue of a box of S . In the following, we suppose given a relation \rightarrow on contexts such that $\rightarrow \subseteq \mapsto$.

Definition 15. A \rightarrow -copy context is a context of the shape $((e, P), [!_t]@T)$ such that for every $u \sqsupseteq t$, there exists a path of the shape $((e, P), [!_u]@T) \rightarrow^* ((_, _), [!_e])$.

Let $(B, P) \in \text{Pot}(B_G)$, the set $\text{Cop}_{\rightarrow}(B, P)$ of \rightarrow -copies of (B, P) is the set of standard signatures t such that $((\sigma(B), P), [!_t])$ is a \rightarrow -copy context.

For example, for any box B and set S such that $B \notin S$, $\text{Cop}_{\mapsto_S}(B, P) = \{e\}$ (because $((\sigma(B), P), [!_t]) \not\mapsto_S$). In Figure 8, we have $\text{Cop}_{\mapsto_{\{C\}}}(C, []) = \{e, 1(e), r(e)\}$ whereas $\text{Cop}_{\mapsto_{\{A, C\}}}(C, []) = \{e, 1(e), r(e), 1(1(e)), 1(r(e)), r(1(e)), r(r(e))\}$.

Definition 16. Let e be an edge of G such that $e \in B_{\partial(e)} \subset \dots \subset B_1$. We define $\text{Can}_{\rightarrow}(e)$ as the set of potentials $[s_1; \dots; s_{\partial(e)}]$ such that:

$$\forall 1 \leq i \leq \partial(x), s_i \in \text{Cop}_{\rightarrow}(B_i, [s_1; \dots; s_{i-1}])$$

For instance, in Figure 8, $\text{Can}_{\mapsto_{\{B\}}}(w) = \{(w, [e; e]), (w, [e; 1(e)]), (w, [e; r(e)])\}$ and $\text{Can}_{\mapsto_{\{C\}}}(d) = \{(d, [e]), (d, [1(e)]), (d, [r(e)])\}$.

We can notice that, in particular, the definitions of $\text{Cop}_{\mapsto}(B, P)$ and $\text{Can}_{\mapsto}(x)$ match respectively the definitions of $\text{Cop}(B, P)$ (Definition 7) and $\text{Can}(x)$ (Definition 8). Finally, we define in Definition 17 a notion of \rightarrow -canonical contexts. Intuitively⁶, every context reachable from $((\sigma(B), P), [!_t])$ by a \rightarrow -path with $(B, P) \in \text{Can}_{\rightarrow}(B)$ and $t \in \text{Cop}_{\rightarrow}(B, P)$, is a \rightarrow -canonical context.

⁶This property is not true for every \rightarrow relation, but is true if \rightarrow is of the shape \mapsto_S .

Definition 17. A \rightarrow -canonical context is a context $((e, [P_1; \dots; P_{\partial(e)}], [T_1; \dots; T_k])$ such that $(e, P) \in \text{Can}_{\rightarrow}(e)$ and:

- For every $T_i = !_t$, $((e, [P_1; \dots; P_{\partial(e)}], [!_t; T_{i+1}; \dots; T_k])$ is a \rightarrow -copy context.
- For every $T_i = ?_t$, $((\bar{e}, [P_1; \dots; P_{\partial(e)}], [!_t; T_{i+1}^\perp; \dots; T_k^\perp])$ is a \rightarrow -copy context.

Let us consider a potential box (B, P) and $t \in \text{Cop}(B, P)$, then there exists a context $((e, Q), [!_e])$ such that $((\sigma(B), P), [!_t]) \mapsto^* ((e, Q), [!_e])$. If some of those \mapsto steps are not in \rightarrow , we may have $((\sigma(B), P), [!_t]) \rightarrow^* ((f, R), [!_v]) \rightarrow$ with $v \neq e$. In this case, t would not be a \rightarrow -copy of (B, P) . However, there exist “truncations” of t which are \rightarrow -copy of (B, P) (at least, e verify those properties).

Definition 18. We define “ t is a truncation of t' ” (written $t \blacktriangleleft t'$) by induction on t . For every signature t, t', u, u' , we set $e \blacktriangleleft t$ and if we suppose $t \blacktriangleleft t'$ and $u \blacktriangleleft u'$ then $l(t) \blacktriangleleft l(t')$, $r(t) \blacktriangleleft r(t')$, $p(t) \blacktriangleleft p(t')$ and $n(t, u) \blacktriangleleft n(t', u')$.

As hinted earlier, we want to define $((\sigma(B), P), [!_t])/\rightarrow$ as the “biggest” \rightarrow -copy u of (B, P) such that $u \blacktriangleleft t$. But we have not precised the meaning of “biggest” yet. The solution we chose is to first maximize the rightmost branch. Then, once this branch is fixed, we maximize the second rightmost branch and so on. Formally, we define “biggest” as “the maximum for the order \preceq ” with \preceq defined as follows.

Definition 19. We first define a strict order \triangleleft on signatures by induction. For every signature t, t', u, v , we set $e \triangleleft t$. And, if we suppose $t \triangleleft t'$, then $l(t) \triangleleft l(t')$, $r(t) \triangleleft r(t')$, $p(t) \triangleleft p(t')$, $n(u, t) \triangleleft n(v, t')$ and $n(t, u) \triangleleft n(t', u)$.

Then we define an order \preceq on signatures by: $t \preceq t'$ iff either $t = t'$ or $t \triangleleft t'$.

Lemma 20 ([26]). Let t be a signature, then \preceq is a total order on $\{u \in \text{Sig} \mid u \blacktriangleleft t\}$.

Thanks to Lemma 20, the set $\text{Restr}_{\rightarrow}((\sigma(B), P), [!_t])$ defined below is totally ordered by \preceq and finite (if t is of size k , it has at most 2^k truncations) so it admits a maximum for \preceq , written $((\sigma(B), P), [!_t])/\rightarrow$.

Definition 21. Let $((e, P), [!_t]@T) \in \text{Cont}_G$, we define $\text{Restr}_{\rightarrow}((e, P), [!_t]@T)$ as the set of signatures u such that $u \blacktriangleleft t$ and $((e, P), [!_u]@T)$ is a \rightarrow -copy context. Then, we define $((e, P), [!_t]@T)/\rightarrow$ as the maximum (for \preceq) element of $\text{Restr}_{\rightarrow}((e, P), [!_t]@T)$.

For example, in the proof-net of Figure 8, $\text{Restr}_{\mapsto_{C_1}}((\sigma(C), []), [!_{l(r(e))}]) = \{e, l(e)\}$ so we have $((\sigma(C), []), [!_{l(r(e))}])/\mapsto_{C_1} = l(e)$.

In Figure 8, $((\sigma_1(A), []), [!_u]) \mapsto_S ((\sigma(A), []), [!_u])$ for any $u \in \text{Sig}$. So, for any $t \in \text{Sig}$, $((\sigma_1(A), []), [!_t])/\mapsto_S = ((\sigma(A), []), [!_e])/\mapsto_S$. Lemma 22 generalizes this observation.

Lemma 22 ([26]). Let $t \in \text{Sig}$. We suppose that, for every $u \blacktriangleleft t$ and $v \sqsupseteq u$, we have $((e, P), [!_v]@T) \rightarrow ((f, Q), [!_v]@U)$. Then, $((e, P), [!_t]@T)/\rightarrow = ((f, Q), [!_t]@U)/\rightarrow$.

Now, for any potential edge (e, P) , we want to define $(e, P)^\rightarrow$ as the “biggest” truncation P' of P such that (e, P') is a \rightarrow -canonical edge. We first maximize the leftmost signature, then the second, and so on.

Definition 23. For every potential edge (e, P) , we define $(e, P)^\rightarrow$ by induction on $\partial(e)$. If $\partial(e) = 0$, then we set $(e, [])^\rightarrow = (e, [])$. Otherwise we have $P = Q.t$, let B be the deepest box containing e , $(\sigma(B), Q') = (\sigma(B), Q)^\rightarrow$ and $t' = ((\sigma(B), Q'), [!_t])^\rightarrow$ then we set $(e, Q.t)^\rightarrow = (e, Q'.t')$.

For example, in the proof-net of Figure 8, $(w, [r(e); l(e)])^{\rightarrow[B]} = (w, [e; l(e)])$.

Definition 24. We extend \blacktriangleleft on Pot by $[p_1; \dots; p_k] \blacktriangleleft [p'_1; \dots; p'_k]$ iff for $1 \leq i \leq k$, $p_i \blacktriangleleft p'_i$.

We can notice that, in the same way as the definition of $Can(e)$ only depends on the boxes containing e (cf. page 13), the definition of $(e, P)^\rightarrow$ only depends on the boxes containing e . We formalize it with the next lemma.

Lemma 25. If $e, f \in \vec{E}_G$ belong to the same boxes, $(e, P)^\rightarrow = (e, P')$ iff $(f, P)^\rightarrow = (f, P')$.

Let us suppose that $((\sigma(B), P), [!_t]) \rightsquigarrow^* ((e, Q), [!_e])$ and S is the set of boxes which are entered by their principal door by this path. Then, we prove that it is enough to know $(e, Q)^{\rightarrow_S}$ to trace back the path (Lemma 30). To do so, we need to prove that for every intermediary step $((e_k, P_k), T_k) \rightsquigarrow ((e_{k+1}, P_{k+1}), T_{k+1})$ we have enough information about P_{k+1} and T_{k+1} to determine e_k . This is the role of the following definition. As an intuition, if $((e, P'), T') = ((e, P), T)^\rightarrow$ then $((e, P'), T')$ is the “biggest” \rightarrow -canonical context which is a truncation of $((e, P), T)$.

Definition 26. For $((e, P), [T_n; \dots; T_1]) \in Cont_G$ we define $((e, P), [T_n; \dots; T_1])^\rightarrow$ as $((e, P'), [T'_n; \dots; T'_1])$ with $(e, P') = (e, P)^\rightarrow$ and T'_i defined by induction on i as follows:

- If $T_i = !_t$, then $T'_i = !_{t'}$ with $t' = ((e, P'), [!_t; T'_{i-1}; \dots; T'_1])^\rightarrow$.
- If $T_i = ?_t$, then $T'_i = ?_{t'}$ with $t' = ((\bar{e}, P'), [!_t; T'^{\perp}_{i-1}; \dots; T'^{\perp}_1])^\rightarrow$.
- Otherwise, $T'_i = T_i$.

Lemma 27 is a generalization of Lemma 22 to contexts. For example, in Figure 8, for every $S \subseteq B_G$ and trace T we have, $((d, [r(n(e, e))]), T) \rightsquigarrow_S^3 ((\bar{h}, []), T.!_{n(e, e)})$ and $((h, []), T^\perp.?_{n(e, e)}) \mapsto_S^3 ((\bar{d}, [r(n(e, e))]), T^\perp)$. So for every $t, u \in Sig$, there exist $v, w \in Sig$ such that

$$\begin{aligned} ((\bar{h}, []), [!_t; ?_u; !_{n(e, e)}])^{\mapsto_S} &= ((\bar{h}, []), [!_v; ?_w; !_{\perp}]) \\ ((d, [r(n(e, e))]), [!_t; ?_u])^{\mapsto_S} &= ((d, [-]), [!_v; ?_w]) \end{aligned}$$

Lemma 27. Let $(e, P), (e, Q)$ be potential edges and U, V be lists of trace elements. Let us suppose that, for every trace element list T , $((e, P), T @ U) \rightarrow ((f, Q), T @ V)$ and $((\bar{f}, Q), T^\perp @ V^\perp) \rightarrow ((\bar{e}, P), T^\perp @ U^\perp)$. Then, for any trace T , $((e, P), T @ U)^\rightarrow$ and $((f, Q), T @ V)^\rightarrow$ are of the shape $(_, T' @ U')$ and $(_, T' @ V')$ with $|T| = |T'|$.

Proof. Let us write $[T_k; \dots; T_1]$ for T , $(_, [T'_k; \dots; T'_1] @ U')$ for $((e, P), T @ U)^\rightarrow$ and $(_, [T''_k; \dots; T''_1] @ V')$ for $((f, Q), T @ V)^\rightarrow$. We prove $T'_i = T''_i$ by induction on i .

If $T_i = !_t$, then we have $T'_i = !_{t'}$ with $t' = ((e, P), [!_t; T'_{i-1}; \dots; T'_1] @ U')^\rightarrow$ and $T''_i = !_{t''}$ with $t'' = ((f, Q), [!_t; T''_{i-1}; \dots; T''_1] @ V')^\rightarrow$. By the induction hypothesis, we have $[T'_{i-1}; \dots; T'_1] = [T''_{i-1}; \dots; T''_1]$. By assumption $((e, P), [!_t; T'_{i-1}; \dots; T'_1] @ U') \rightarrow^* ((f, Q), [!_t; T'_{i-1}; \dots; T'_1] @ V')$. Thus, by Lemma 22, $t'_i = t''_i$ so $T'_i = T''_i$.

The case $T_i = ?_t$ is similar (using the $((\bar{f}, Q), T^\perp @ V^\perp) \rightarrow ((\bar{e}, P), T^\perp @ U^\perp)$ hypothesis). \square

3.4. Elementary bound for \rightsquigarrow -stratified proof-nets

We consider the following theorem as the main technical innovation of this paper. It uses the notions of the previous section to trace back \rightsquigarrow -paths. In order to bound W_G , we need to bound the number of copies of potential boxes. The usual way to prove the elementary bound on LLL is a round-by-round cut-elimination procedure: we first reduce every cut at depth 0. Because of the absence of dereliction in ELL , none of these step creates new cuts at depth 0. So this round terminates in at most $|E_G|$ steps. Because each step may at most double the size of the proof-net, the size of the proof-net at the end of round 0 is at most $2^{|E_G|}$. Then we reduce the cuts at depth 1, because of the previous bound there at most $2^{|E_G|}$ such cuts, and the reduction of those cuts does not create any new cut...

The original proof of the elementary bound of L^3 relies on a similar round-by-round procedure which is more complex because reducing a cut at level i can create new cuts at level i , and a box of level i can be contained in a box of higher level. While Dal Lago adapted to context semantics the round-by-round procedure of ELL concisely in [8], the round-by-round procedure of L^3 was only adapted to context semantics by Perrinel [24] (a work which is the basis of this article).

Theorem allows us to bring round-by-round procedures where strata differ from depth, to context semantics. We explained that $(e, P)^{\mapsto s}$ corresponds to a residue e' of e , such that we only fired cuts involving principal door of boxes of S . In a round-by-round procedure, after the i -th round we have a bound on the number of such $(e, P)^{\mapsto s}$. By tracing back a path from $((e, P), [!_e])$ until a potential box (B, P) using only the information $(e, P)^{\mapsto s}$, we show that there is only one residue of e' which will be cut with B (more precisely its residue corresponding to (B, P)). This allows us to prove a bound on the number of copies of (B, P) .

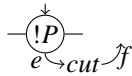
While we will use other criteria and technical results to deal with the \hookrightarrow steps, both the proofs of the elementary bound and the proofs of the polynomial bound rely on Theorem 28.

Theorem 28. *Let G be a proof-net and $S \subset B_G$. Let C_e , C_f and C'_f be contexts such that $C_e \rightsquigarrow_S C_f$ and $C_f^{\mapsto s} = C'^{\mapsto s}_f$, then there exists a context C'_e such that $C'_e \rightsquigarrow_S C'_f$ and $C_e^{\mapsto s} = C'^{\mapsto s}_e$.*

Proof. We detail an easy step (crossing a \mathfrak{A} node upward). Most of the other steps are quite similar. For the steps which offer some particular difficulty, we only detail the points which differ from crossing a \mathfrak{A} upward.

Let us suppose that $C_e = ((e, P), T \cdot \otimes_l) \rightsquigarrow_S ((f, P), T) = C_f$ (crossing a \mathfrak{A} upwards, such that \bar{f} is not a principal edge) and $C_f^{\mapsto s} = C'^{\mapsto s}_f$. So C'_f is of the shape $((f, P'), T')$. We set $C'_e = ((e, P'), T' \cdot \otimes_l)$. Let $((f, P''), T'') = C'^{\mapsto s}_f$, then $(f, P)^{\mapsto s} = (f, P')^{\mapsto s} = (f, P'')$. So, by Lemma 25, $(e, P)^{\mapsto s} = (e, P')^{\mapsto s} = (e, P'')$. Moreover, by Lemma 27, $C_e^{\mapsto s} = ((e, P''), T'' \cdot \otimes_l)$ and $C'^{\mapsto s}_e = ((e, P''), T'' \cdot \otimes_l)$ so $C_e^{\mapsto s} = C'^{\mapsto s}_e$.

Let us consider the case where e is the principal edge of a box B (we consider the case where we cross a *cut*), we suppose that we



have $C_e = ((e, P), T.!_t) \rightsquigarrow_S ((f, P), T.!_t) = C_f$ ⁷. So C'_f is of the shape $((f, P'), T'.!_{t'})$. We set $C'_e = ((e, P'), T'.!_{t'})$. By supposition, $C_f \mapsto_S C'_f \mapsto_S ((f, P''), T''.!_{t''})$. In particular $((f, P''), [!_t]) \mapsto_S = ((f, P''), [!_{t'}]) \mapsto_S$. If $B \in S$, by Lemma 27, we have $C_e \mapsto_S = C'_e \mapsto_S = ((e, P''), T''.!_{t''})$. Otherwise, we have $C_e \mapsto_S = C'_e \mapsto_S = ((e, P''), T''.!_e)$.

Let us consider the case where \bar{f} is the principal edge of a box B (we consider the case where we cross a *cut*) with $C_e = ((e, P), T.?_t) \rightsquigarrow_S ((f, P), T.?_t) = C_f$. So C'_f is of the shape $((f, P'), T'.?_{t'})$. We set $C'_e = ((e, P'), T'.?_{t'})$. By supposition, $C_f \mapsto_S = C'_f \mapsto_S ((f, P''), T''.?_{t''})$. By definition of \rightsquigarrow_S , B is in S . So, we can notice that $C'_e \rightsquigarrow_S C'_f$ and, using Lemma 27, we have $C_e \mapsto_S = C'_e \mapsto_S = ((e, P''), T''.?_{t''})$.

Let us suppose that $C_e = ((e, P), T.?_t) \rightsquigarrow_S ((f, P, t), T) = C_f$ (crossing the principal door of C upwards). Then, C'_f must be of the shape $((f, P'.t'), T')$. We set $C'_e = ((e, P'), T'.?_{t'})$. The only particular point is to prove that $((\bar{e}, P) \mapsto_S, [!_t]) \mapsto_S = ((\bar{e}, P') \mapsto_S, [!_{t'}]) \mapsto_S$. By definition, $(f, P.t) \mapsto_S = (f, Q.u)$ with $(\bar{e}, P) \mapsto_S = (\bar{e}, Q)$ and $(\bar{e}, Q), [!_t] \mapsto_S = u$. Similarly, $(f, P'.t') \mapsto_S = (f, Q'.u')$ with $(\bar{e}, P') \mapsto_S = (\bar{e}, Q')$ and $(\bar{e}, Q'), [!_{t'}] \mapsto_S = u'$. We know that $C_f \mapsto_S = C'_f \mapsto_S$, so $(f, Q.t) \mapsto_S = (f, Q'.t') \mapsto_S$. Thus $u = u'$, i.e. $((\bar{e}, P) \mapsto_S, [!_t]) \mapsto_S = ((\bar{e}, P') \mapsto_S, [!_{t'}]) \mapsto_S$.

Let us suppose that $C_e = ((e, P, t), T) \rightsquigarrow_S ((f, P), T.!_t) = C_f$ (crossing the principal door of B downwards). Then C'_f must be of the shape $((f, P'), T'.!_{t'})$. We set $C'_e = ((e, P'.t'), T')$. The only particular point is to prove that $(e, P.t) \mapsto_S = (e, P'.t') \mapsto_S$. By definition of $(_, _) \mapsto_S$, $(e, P.t) \mapsto_S = (e, Q.u)$ with $(f, P) \mapsto_S = (f, Q)$ and $((f, Q), [!_t]) \mapsto_S = u$. Similarly, $(e, P'.t') \mapsto_S = (e, Q'.u')$ with $(f, P') \mapsto_S = (f, Q')$ and $((f, Q'), [!_{t'}]) \mapsto_S = u'$. By supposition,

$$\begin{aligned} ((f, P), T.!_t) \mapsto_S &= ((f, P'), T'.!_{t'}) \mapsto_S \\ \left((f, P) \mapsto_S, -@ \left[!_{((f, P) \mapsto_S, [!_t]) \mapsto_S} \right] \right) &= \left((f, P') \mapsto_S, -@ \left[!_{((f, P') \mapsto_S, [!_{t'}) \mapsto_S} \right] \right) \\ ((f, Q), -@[!_u]) &= ((f, Q'), -@[!_{u'}]) \\ Q.u &= Q'.u' \end{aligned}$$

The steps crossing auxiliary doors are similar to the steps crossing principal doors (dealt with above). To deal with the $?C$ node, one has to notice that $t \sqsubseteq u$ iff $1(t) \sqsubseteq 1(u)$. The steps crossing $?N$ nodes are quite technical, but do not bring any insight on the result. Those steps are described in [26]. \square

Theorem 28 allows us to trace back some \rightsquigarrow paths provided that we have some information about the last context of the path. In this subsection, we show how this implies an elementary bound (Lemma 35). But, first, we need some technical lemmas.

Lemma 29 ([26]). *Let $\rightarrow \subseteq \mapsto$. If $((\sigma(B), P), [!_t]) \rightarrow^* C$, then there exists a unique context $((\sigma(B'), P'), [!_{t'}])$ such that $((\sigma(B), P), [!_t]) \rightarrow^* ((\sigma(B'), P'), [!_{t'}]) (\rightsquigarrow \cap \rightarrow)^* C$.*

⁷If the rightmost trace element of C_e is not of the shape $!_t$, the proof does not offer any additional difficulty compared to the step presented above.

Let G be a \rightsquigarrow -stratified proof-net and $n \in \mathbb{N}$, we set $S_n = \{B \in B_G \mid s_{\rightsquigarrow}(B) \leq n\}$. Let us notice that, if $s_{\rightsquigarrow}(B) \leq n$, the set of boxes C such that $B \rightsquigarrow C$ is included in S_{n-1} . So, we will be able (thanks to Lemma 28) to bound the number of copies of boxes of S_n depending on the maximum number of copies of boxes of S_{n-1} . This corresponds to the round-by-round cut-elimination procedure used to prove the bounds on ELL , LLL , L^3 and L^4 .

To make notations readable, we write \mapsto_n for \mapsto_{S_n} , \rightsquigarrow_n for \rightsquigarrow_{S_n} , $((e, P), T)^n$ for $((e, P), T)^{\mapsto_{S_n}}$, $Cop_n(B, P)$ for $Cop_{\mapsto_{S_n}}(B, P)$ and so on.

Lemma 30. *Let $n \in \mathbb{N}$. If $((\sigma(B), P), [!_t]) \mapsto_n C_k \cdots \mapsto_n C_0$ and $C_0^{n-1} = C_0'^{n-1}$ then there exists $(C'_i)_{0 \leq i \leq k}$ such that $C'_k \mapsto_n \cdots \mapsto_n C'_0$ and, for $0 \leq i \leq k$, $C_i^{n-1} = C_i'^{n-1}$.*

Proof. We prove (by induction on i) the existence of a context C'_i such that $C'_i \mapsto_n C'_{i-1}$ and $C_i^{n-1} = C_i'^{n-1}$. If $i = 0$, C'_0 satisfies the property by assumption. Otherwise, by induction hypothesis we know that there exists a context C'_{i-1} such that $(C_{i-1})^{n-1} = (C'_{i-1})^{n-1}$.

If the $C_i \mapsto_n C_{i-1}$ step is a \hookrightarrow step, it is of the shape $C_i = ((\sigma_j(D), Q), [!_u]) \hookrightarrow ((\sigma(D), Q), [!_u]) = C_{i-1}$. So C'_{i-1} is of the shape $((\sigma(D), Q'), [!_{u'}])$ with $(\sigma(D), Q)^{n-1} = (\sigma(D), Q')^{n-1} = (\sigma(D), Q'')$ and $((\sigma(D), Q'), [!_{u'}])^{n-1} = ((\sigma(D), Q''), [!_{u'}])^{n-1} = u''$. Let us set $C'_i = ((\sigma_j(D), Q'), [!_{u'}])$. By Lemma 25, $(\sigma_j(D), Q)^{n-1} = (\sigma_j(D), Q')^{n-1} = (\sigma_j(D), Q'')$. By Lemma 22, $((\sigma_j(D), Q'), [!_{u'}])^{\mapsto_s} = ((\sigma_j(D), Q''), [!_{u'}])^{\mapsto_s} = u''$. So $C_i^{\mapsto_s} = C_i'^{\mapsto_s} = ((\sigma_j(D), Q''), [!_{u'}])$.

Otherwise, $C_i \rightsquigarrow C_{i-1}$ step so there exists a context of the shape $((\sigma(D), Q), [!_u])$ such that $((\sigma(B), P), [!_t]) \mapsto_n ((\sigma(D), Q), [!_u])(\rightsquigarrow \cap \mapsto_n)^+ C_{i-1}$ (Lemma 29). And, by definition of \mapsto_n , $D \in S_n$. We prove that the last step of the path is in \rightsquigarrow_{n-1} . We suppose C_{i-1} is of the shape $((\sigma(D_i), Q_i), [!_v])$ (otherwise, it is immediate by definition of \rightsquigarrow_{n-1}). We can notice that $D \rightsquigarrow D_i$ so $s_{\rightsquigarrow}(D_i) < s_{\rightsquigarrow}(D) \leq n$, which means that $D_i \in S_{n-1}$. Thus, we have $C_i \rightsquigarrow_{n-1} C_{i-1}$. By Theorem 28, there exists a context C'_i such that $C'_i \rightsquigarrow_{n-1} C'_{i-1}$ and $C_i^{n-1} = C_i'^{n-1}$. \square

Lemma 31 ([26]). *Let $S \subseteq B_G$. If $((e, P), [!_t]@T) \mapsto_S^* ((f, Q), [!_u]@U)$, for every $u' \in Sig$, there exists $t' \in Sig$ such that $((e, P), [!_{t'}]@T) \mapsto_S^* ((f, Q), [!_{u'}]@U)$.*

Proof. It is enough to prove it for one step. We can examine every possible step, each case is straightforward: the steps sometimes depend on t , never on u . For instance, let us suppose that $((e, P), [!_{1(u)}]) \mapsto_S ((f, Q), [!_u])$ (crossing a $?C$ upwards). Then, for every $u' \in Sig$, we have $((e, P), [!_{1(u')}]) \mapsto_S ((f, Q), [!_{u'}])$ so we can set $t' = 1(u')$. \square

Lemma 32 (strong acyclicity). *Let G be a normalizing proof-net. For every $n \in \mathbb{N}$, if $((\sigma(B), P), [!_t]) \mapsto_n^* ((e, Q), [!_u]) \mapsto_n^+ ((e, Q'), [!_v])$ then $(e, Q)^{n-1} \neq (e, Q')^{n-1}$.*

Proof. We prove it by contradiction. We suppose that $((\sigma(B), P), [!_t]) \mapsto_n^l ((e, Q), [!_u])$ and $((\sigma(B), P), [!_t]) \mapsto_{S_n}^{l+m} ((e, Q'), [!_{u'}]) = D'$, and $(e, Q)^{n-1} = (e, Q')^{n-1}$. Then, $((e, Q), [!_{u'}])^{n-1} = D'^{n-1}$. By Lemma 30, there exists a context C'_1 such that $C'_1 \mapsto^{l+m} ((e, Q), [!_{u'}])$ and $C'_1{}^{n-1} = ((\sigma(B), P), [!_t])^{n-1}$. So C'_1 is of the shape $((\sigma(B), P_1), [!_{t'}])$. By Lemma 31, there exists a signature t_1 such that $((\sigma(B), P_1), [!_{t_1}]) \mapsto^{l+m} ((e, Q), [!_u])$ so $((\sigma(B), P_1), [!_{t_1}]) \mapsto^{l+2m} ((e, Q'), [!_{u'}])$.

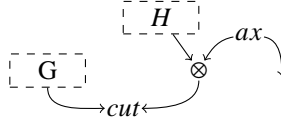


Figure 9: This proof-net, written $(G)H$, corresponds to the application of a function G to an argument H .

We define C_1 as the context $((\sigma(B), P_1), [!_t])$. For $k \in \mathbb{N}$, we can define by induction on k a context $C_k = ((\sigma(B), P_k), [!_{t_k}])$ such that $C_k \mapsto_n^{l+k \cdot m} D$ and $C_k \mapsto_n^{l+(k+1) \cdot m} D'$.

Thus, if $m > 0$, we define an infinite path. In particular, this path goes through infinitely many contexts of shape $((\sigma(B), P'), [!_{t'}])$. According to Corollary 11, the number of canonical potentials for an edge is finite. So there is some $(\sigma(B), P') \in \text{Can}(\vec{E}_G)$ and $v, v' \in \text{Sig}$ such that $((\sigma(B), P'), [!_v]) \mapsto^+ ((\sigma(B), P'), [!_{v'}])$. This is impossible because normalizing proof-nets are acyclic (Lemma 12). This is a contradiction, so our hypothesis is wrong, $m = 0$. There is no path of the shape $((\sigma(B), P), [!_t]) \mapsto_n^* ((e, Q), [!_u]) \mapsto_n^+ ((e, Q'), [!_v])$ with $(e, Q)^{n-1} = (e, Q')^{n-1}$. \square

Lemma 33 ([26]). *The number of signatures whose depth is $\leq d$ is at most 2^{2^d}*

Lemma 34. *If $\left| \left\{ (e, Q)^{n-1} \mid \exists t, u \in \text{Sig}, ((\sigma(B), P), [!_t]) \mapsto_n^* ((e, Q), [!_u]) \right\} \right| \leq M$, then $|\text{Cop}_n(B, P)|$ is bounded by 2^{2^M} .*

Proof. Let us consider $u \in \text{Sig}$ such that there exists $t \in \text{Cop}_n(B, P)$ such that $t \sqsubseteq u$. By definition of $\text{Cop}_n(_)$ (Definition 15, in page 18), there exists a path of the shape $((\sigma(B), P), [!_u]) \mapsto_n^* ((_, _), [!_e])$. We consider u as a tree. During the path beginning by $((\sigma(B), P), [!_u])$, the height of the left-most branch of u (viewed as a tree) decreases to 0 (the height of e). The height of the left-most branch decreases only by crossing a $?C$ or $?N$ nodes upwards (which corresponds to contexts of the shape $((e, Q), [!_v])$) and during those steps it decreases by exactly 1. So the height of the left-most branch of u is inferior to the number of instances of contexts of the shape $((e, Q), [!_v])$ through which the path goes. From Lemma 32, each $(e, Q)^{n-1}$ is represented at most once. So the height of the left-most branch of u is inferior to M .

Let t be a \mapsto_n -copy of (B, P) , then the height of t is the height of its deepest branch. Once we consider signatures as trees, a simplification u of t can be viewed as a subtree of t obtained as follows: we choose a branch of t and u is the part of t on the right of this branch, in particular this branch becomes the leftmost branch of u . So there exists a simplification u of t such that the leftmost branch of u is the deepest branch of t . So the height of t is equal to the height of the leftmost branch of u . By the preceding paragraph, the height of the leftmost branch of u is at most M so the height of t is at most M . The result is obtained by Lemma 33. \square

In order to express elementary bounds, we define the notation 2_n^x (with $n \in \mathbb{N}$ and $x \in \mathbb{R}$) by induction on n : $2_0^x = x$ and $2_{n+1}^x = 2_n^{2^x}$. So 2_n^x is a tower of exponentials of height n with top exponent x .

Theorem 35. *If a proof-net G normalizes and is \rightsquigarrow -stratified, then the length of its longest path of reduction is bounded by $2^{\frac{|\vec{E}_G|}{3}}$.*

Proof. By Lemma 34 and definition of $Can_{n-1}(_)$ we have:

$$\begin{aligned} \max_{(B,P) \in Pot(B_G)} |Cop_n(B, P)| &\leq 2^{2^2 |Can_{n-1}(\vec{E}_G)|} \\ \max_{e \in \vec{E}_G} |Can_n(e)| &\leq \left(2^{2^2 |Can_{n-1}(\vec{E}_G)|} \right)^{\partial_G} \\ |Can_n(\vec{E}_G)| &\leq |\vec{E}_G| \left(2^{\partial_G \cdot 2^2 |Can_{n-1}(\vec{E}_G)|} \right) \end{aligned}$$

We define u_n as $2^{\frac{|\vec{E}_G|}{3 \cdot n}}$. We show by induction that, for every $n \in \mathbb{N}$, $|Can_n(\vec{E}_G)| \leq u_n$. For $n = 0$, we can notice that for every $e \in \vec{E}_G$, we have $|Can_0(e)| = 1$ (the only canonical potentials are lists of e) so $|Can_0(\vec{E}_G)| \leq |\vec{E}_G| \leq u_0$. If $n \geq 0$, let us notice that G has at least two boxes so $|\vec{E}_G| \geq 4$. We have the following inequalities (to simplify the equations, we write s for $|\vec{E}_G|$):

$$\begin{aligned} |Can_{n+1}(\vec{E}_G)| &\leq s \left(2^{\partial_G \cdot 2^2 |Can_n(\vec{E}_G)|} \right) \leq s \left(2^{\partial_G \cdot 2^{2 \cdot u_n}} \right) \leq 2^{\frac{s}{2}} \left(2^{s \cdot 2^{2 \cdot u_n}} \right) \\ \log \left(|Can_{n+1}(\vec{E}_G)| \right) &\leq \frac{s}{2} + s \cdot 2^{2 \cdot u_n} \leq (2 \cdot s) \cdot 2^{2 \cdot u_n} \leq 2^{s+2 \cdot u_n} \leq 2^{4u_n} \leq 2^{2u_n} \\ |Can_{n+1}(\vec{E}_G)| &\leq 2^{\frac{u_n}{3}} = 2^{\frac{s}{3n+3}} = u_{n+1} \end{aligned}$$

Then, Theorem 11 gives us the announced bound. \square

Let us consider the application of a proof-net G to H (Figure 9). If \rightsquigarrow is acyclic on $(G)H$, then $|\rightsquigarrow| \leq |B_{(G)H}| \leq |B_G| + |B_H|$. It is reasonable⁸ to assume that the number of boxes does not depend on the argument of the function. So, by Theorem 35, the length of the normalization sequence is bounded by $e_G(x)$ with x the size of the argument and e_G an elementary function which does not depend on the argument.

4. Paths criteria for polynomial time

4.1. Dependence control

Though \rightsquigarrow -stratification gives us a bound on the length of the reduction, elementary time is not considered as a reasonable bound, as it rises extremely fast with the size of the input. Cobham-Edmons thesis asserts that *Ptime* corresponds to feasible problems. It suffers some limits:

⁸More details at the end of Section 4.2.

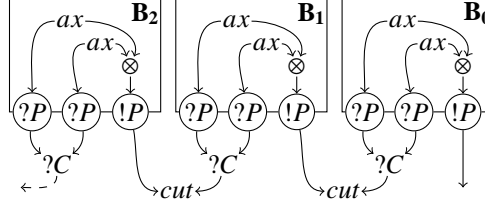


Figure 10: This proof-net (if extended to n boxes) reduces in $O(2^n)$ reduction steps

- When one is only interested in very small inputs, the asymptotical complexity is not a concern
- It does not account for constants and exponents.

However, in practice, the programs which we consider tractable mostly correspond to programs enjoying a polynomial bound on their time complexity. This is why we look for criteria entailing a polynomial bound on W_G . Figure 10 shows us a way for the complexity to arise despite \rightsquigarrow -stratification. On this proof-net, B_1 has two residues. Each residue of B_1 creates two residues of B_2 (so 4 residues in total). If we extend this sequence of boxes, B_n has at least 2^n residues. From a context semantics perspective, $|Cop(B_i, [])|$ depends non-additively on $|Cop(B_{i-1}, [])|$. Indeed, for any $t \in Cop(B_{i-1}, [])$, both $l(t)$ and $r(t)$ are in $Cop(B_i, [])$. Thus, for every copy in B_0 there exist at least 2^i copies of B_i .

This proof-net is similar to the λ -term $(\lambda x.\langle x, x \rangle) \cdots (\lambda x.\langle x, x \rangle)y$ (in λ -calculus with pairs) which reduces to a λ -term of size $O(2^n)$ (with n the number of successive applications of $\lambda x.\langle x, x \rangle$). Let us observe that the number of \rightarrow_β steps depends on the strategy: call-by-name normalizes in $\Theta(2^n)$ steps while call-by-value normalizes in $\Theta(n)$ steps (but, because the term size grows exponentially, the execution time is in $\Theta(2^n)$ independently of the reduction strategy). The exponential blow-up happens because there are two free occurrences of x in $\lambda x.\langle x, x \rangle$ (this corresponds in Figure 10 to the two auxiliary doors by box which come from the same contraction node).

In [27], this situation is called a chain of *spindles*. We call *dependence control condition* any restriction on linear logic which aims to tackle this kind of spindle chains. The dependence control in *LLL* [15] is to limit the number of auxiliary doors of each $!P$ -box to 1. The dependence control in *SLL* [20] is to forbid auxiliary doors above contraction nodes.

However, those conditions forbid many proof-nets normalizing in polynomial time. For instance, the proof-net of Figure 11 normalizes in linear time, even if the boxes have two auxiliary doors one of which is above a $?C$ node. The copies of C_i depend on the copies of C_{i-1} because $Cop(C_i, []) = \{e, r(e)\} \cup \{l(t) \mid t \in Cop(C_{i-1}, [])\}$. But the dependence is additive: $|Cop(C_i, [])| = 2 + |Cop(C_{i-1}, [])|$.

In terms of context semantics, to give a bound on the number of copies of a potential box, we want to trace back a path $((\sigma(B), P), [!_l]) \mapsto^* ((e, Q), [!_e])$ with as little information on the path as possible. Theorem 28 (and the injectivity of \rightsquigarrow) allows us to trace back \rightsquigarrow steps. However, we need additional information to trace back \hookrightarrow steps

because \hookrightarrow is not injective. For instance, in Figure 10, we have:

$$\begin{aligned} ((\sigma(B_2), []), [!_{1(e)}]) \rightsquigarrow^2 C_e &= ((\overline{\sigma_1(B_1)}), [], [!_e]) \hookrightarrow ((\sigma(B_1), []), [!_e]) = C_f \\ ((\sigma(B_2), []), [!_{r(e)}]) \rightsquigarrow^2 C'_e &= ((\overline{\sigma_2(B_1)}), [], [!_e]) \hookrightarrow ((\sigma(B_1), []), [!_e]) = C_f \end{aligned}$$

Let us consider a $((\sigma(B), P), [!_t]) (\rightsquigarrow_S \cup \hookrightarrow)^* ((e, Q), [!_e])$ path. Thanks to Theorem 28 and Lemma 36, we can trace it back (determine every edge of the path) provided we know $((e, Q), [!_e]) \mapsto^s$ and, for every $((\sigma_i(C), R), [!_u]) \hookrightarrow ((\sigma(C), R), [!_u])$ step of the path, we know i .

Lemma 36. *Let S be a subset of boxes. We suppose that $C_e = ((\overline{\sigma_i(B)}), P), [!_t]) \hookrightarrow C_f$, $C'_e = ((\overline{\sigma_i(B)}), P'), [!_{t'}]) \hookrightarrow C'_f$ and $C_f \mapsto^s = C'_f \mapsto^s$ then $C_e \mapsto^s = C'_e \mapsto^s$.*

Proof. Quite similar to the proof of Theorem 28 (cf. the study of the \mathfrak{A} case). \square

A dependence control condition is a criterion on proof-nets entailing a bound on the number of \hookrightarrow steps for which we need to know the auxiliary edge to be able to trace back a \mapsto -path. Instead of a syntactic criterion (like the ones of the type-systems LLL and SLL), we propose here a semantic criterion on proof-nets. As in Section 3, the criterion is defined as the acyclicity of a relation (written \succ) on boxes. Our criterion is more general than previous systems: every proof-net of (the multiplicative fragments of) LLL , SLL and every *Ptime* sound system of MS satisfies our dependence control condition.

Intuitively $B \succ B'$ means that residues B_1 and B_2 of B are cut, along reduction, with two distinct auxiliary doors $(\sigma_i(_)$ and $\sigma_j(_)$ of residues $(B'_1$ and $B'_2)$ of C . From a context semantics point of view, it corresponds to the existence of \mapsto -paths from the principal door of B to two distinct auxiliary doors of B' .

Let us observe that the relation \succ is defined by considering \mapsto -paths ending by a context on an (reversed) auxiliary edges of a box B' while the relation \rightsquigarrow (Definition 13 in page 17) was defined by considering \rightsquigarrow -paths passing through the (reversed) principal edge of a box B' .

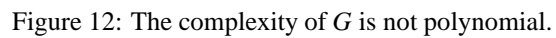
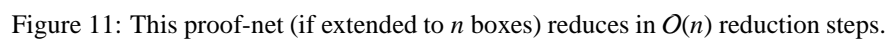
Definition 37. *We set $B \succ B'$ iff there exist $i \neq j$ and paths of the shape:*

$$\begin{aligned} ((\sigma(B), P), [!_t]) \mapsto^+ ((\overline{\sigma_i(B')}), P'_1), [!_e]) \\ ((\sigma(B), P), [!_u]) \mapsto^+ ((\overline{\sigma_j(B')}), P'_2), [!_e]) \end{aligned}$$

In Figure 10, we have $B_i \succ B_{i-1}$ because $((\sigma(B_i), []), [!_{1(e)}]) \mapsto^2 ((\overline{\sigma_1(B_{i-1})}), [], [!_e])$ and $((\sigma(B_i), []), [!_{r(e)}]) \mapsto^2 ((\overline{\sigma_2(B_{i-1})}), [], [!_e])$. Similarly, the proof-net of Figure 12 is not \succ -stratified because $B \succ B$. On the contrary, in Figure 11, $\succ = \emptyset$.

Lemma 38. *Let G be a \rightsquigarrow -stratified proof-net, $s \in \mathbb{N}$ and (B, P) be a potential box with $d = s \succ (B)$. There are at most $\left| \text{Can}_{s-1}(\vec{E}_G) \right|^d$ sequences $(e_i)_{1 \leq i \leq d}$ of directed edges such that, there exists a potential sequence $(P_i)_{1 \leq i \leq d}$, a trace sequence $(T_i)_{1 \leq i \leq d}$ and $t \in \text{Sig}$ such that:*

$$((\sigma(B), P), [!_t]) \mapsto_s ((e_1, P_1), T_1) \mapsto_s \cdots \mapsto_s ((e_{l-1}, P_{l-1}), T_{l-1}) \mapsto_s ((e_l, P_l), [!_e])$$



Proof. We prove it by induction on d . We suppose that there exists a path of the shape $((\sigma(B), P), [!_i]) \mapsto_s ((e_1, P_1), T_1) \mapsto_s \dots \mapsto_s ((e_{l-1}, P_{l-1}), T_{l-1}) \mapsto_s ((e_l, P_l), [!_e])$. If there is a context in the path of the shape $((\sigma(C), Q), [!_j])$ with $s \succ (C) < s \succ (B)$, we set k as the smallest index such that $((e_{k+1}, P_{k+1}), T_{k+1})$ is such a context. Otherwise, we set $k = l$.

First, let us notice that by induction hypothesis, there are at most $|Can_{s-1}(\vec{E}_G)|^{d-1}$ possibilities for e_{k+1}, \dots, e_l . Then, let us determine the number of possibilities for e_1, \dots, e_k . There are at most $|Can_{s-1}(\vec{E}_G)|$ choices for $(e_k, P_k)^{s-1}$. Once $(e_k, P_k)^{s-1}$ is determined, we will prove by contradiction that it determines e_1, \dots, e_k . Let us suppose that there exists two possible sequences: e_1, \dots, e_k and e'_1, \dots, e'_k . Then we consider the lowest j such that $((e_{k-j}, P_{k-j}), T_{k-j})^{s-1} \neq ((e'_{k-j}, P'_{k-j}), T'_{k-j})^{s-1}$. By assumption we have $k > 0$ and, by Theorem 28, the “ $k - j$ and $k' - j$ steps” must be \hookrightarrow steps:

$$\begin{aligned} C_{k-j} &= ((\overline{\sigma_{i_1}(D)}, P_{k-j}), [!_v]) \hookrightarrow ((\sigma(D), P_{k-j}), [!_v]) = C_{k+1-j} \\ C'_{k-j} &= ((\overline{\sigma_{i_2}(D)}, P'_{k-j}), [!_{v'}]) \hookrightarrow ((\sigma(D), P'_{k-j}), [!_{v'}]) = C'_{k+1-j} \end{aligned}$$

with $(C_{k+1-j})^{s-1} = (C'_{k+1-j})^{s-1}$ and $(C_{k-j})^{s-1} \neq (C'_{k-j})^{s-1}$. By Lemma 25, the difference is not on the potential and by Lemma 22 the difference is not on the trace, so the difference is on the edge: $i_1 \neq i_2$. By definition of \succ , it means that $B \succ D$ and $s \succ (D) < s \succ (B)$. This contradicts the definition of k . So our hypothesis is false: if we fix $(e_k, P_k)^{s-1} = (e'_k, P'_k)^{s-1}$, then $[e_1; \dots; e_k] = [e'_1; \dots; e'_k]$.

Thus, we proved that there are at most $|Can_{s-1}(\vec{E}_G)|$ possibilities for e_1, \dots, e_k and at most $|Can_{s-1}(\vec{E}_G)|^{d-1}$ possibilities for e_{k+1}, \dots, e_l . In total, there are at most $|Can_{s-1}(\vec{E}_G)|^d$ possibilities for e_1, \dots, e_l . \square

4.2. Nesting

Lemma 38 bounds the number of paths corresponding to copies, provided that \rightsquigarrow and \succ are acyclic. In the absence of $?N$ nodes, a copy t of (B, P) only contains $l(-)$, $r(-)$ and e constructions. One can reconstruct t by observing the list of contexts in the path, of the shape $((e_i, P_i), [!_i])$ with $\overline{e_i}$ being a premise of a contraction node. This is entirely determined by the sequence e_1, \dots, e_l of edges of the path $((\sigma(B), P), [!_i]) \mapsto ((e_1, -), -) \mapsto \dots \mapsto ((e_l, -), [!_e])$. Thus, if there is no $?N$ node, Lemma 38 bounds the number of copies of (B, P) .

To understand why the $?N$ nodes break this property, we can consider an example in Figure 13. We can notice that \rightsquigarrow and \succ are both the empty relation so $s_{\rightsquigarrow}(B_2) = s_{\rightsquigarrow}(B_1) = s_{\rightsquigarrow}(B_0) = 1$ and $s_{\succ}(B_2) = s_{\succ}(B_1) = s_{\succ}(B_0) = 1$. However, if extended to n boxes, $|Cop(B_n, [])| \geq 3^n$ and the number of \rightarrow_{cut} steps is not polynomial in n .

To guide intuition, we can study a similar situation in λ -calculus with pairs. The λ -term $(\lambda x.(\lambda y. \langle y, y \rangle)x) \dots (\lambda x.(\lambda y. \langle y, y \rangle)x)z$ reduces to a λ -term of size $O(2^n)$ (with n the number of successive applications of $\lambda x.(\lambda y. \langle y, y \rangle)x$). In this case x has only one free occurrence in $\lambda x.(\lambda y. \langle y, y \rangle)x$ (it corresponds to the fact that there is only one auxiliary door in the boxes of Figure 13) however x is duplicated inside $\lambda x.(\lambda y. \langle y, y \rangle)x$ (this term reduces to $\lambda x. \langle x, x \rangle$). This corresponds to the $?C$ node inside the boxes B_i of Figure 13, which duplicates the box B_{i-1} . This is possible because the box B_{i-1} gets inside the box B_i because of the $?N$ node.

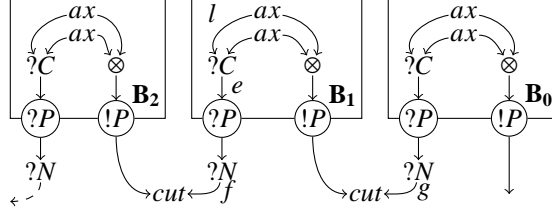


Figure 13: This proof-net (if extended to n boxes) reduces in $O(2^n)$ reduction steps.

We call *nesting* any restriction on linear logic which aims to tackle this kind of chains. The nesting in *LLL* [15], *SLLL* [20], *mL*⁴ [3] and *MS* [27] is the absence of $?N$ node. Lemma 38 states that there are at most $|\vec{E}_G|$ sequences of edges corresponding to copies of $(B_2, [])$, the sequence being entirely determined by the last edge⁹. For instance, knowing that $((\sigma(B_2), []), [!_t]) \mapsto_1^* ((\bar{l}, [p]), [!_e])$ is enough to deduce that:

$$((\sigma(B_2), []), [!_t]) \mapsto_1 ((\bar{f}, []), [!_t]) \mapsto_1 ((\overline{\sigma_1(B_1)}, []), [!_{1(e)}; !_p]) \mapsto_1^2 ((\bar{l}, [p]), [!_e])$$

Thus, we can deduce that t is of the shape $n(1(e), p)$, but we do not know t entirely because p can be any element of $Cop(B_1, []) = \{e, n(e, e), n(1(e), e), n(r(e), e)\}$.

Following the paths *backwards* we can observe that the most important step is $((\bar{f}, []), [!_{n(e, n(p, e))}]) \mapsto_1 ((\overline{\sigma_1(B)}, []), [!_e; !_{n(p, e)}])$ where a difference on the second trace element (which comes from B_1 with $s_{\rightarrow}(B_1) = 1$) becomes a difference on the first trace element, which corresponds to t . The paths corresponding to $n(e, n(1(e), e))$ and $n(e, n(r(e), e))$ are the same, but the paths corresponding to their simplifications are different.

The dependence of $|Cop(B_2, [])|$ on $|Cop(B_1, [])|$ in Figure 13 is similar to the dependence in Figure 10. We define a relation \prec on boxes capturing this dependence. Intuitively $B \prec C$ means that B is cut with a $?N$ node along reduction and the outer residue B_e of B is cut with an auxiliary door of C . The acyclicity of \prec is a nesting condition.

Definition 39. We set $B \prec C$ if there exists a non-standard signature t and a path of the shape:

$$((\sigma(B), P), [!_t]) \mapsto^+ ((\sigma(C), Q), [!_e])$$

For example, in Figure 13, we have $B_2 \prec B_1$ because $p(e)$ is non-standard and $((\sigma(B_2), []), [!_{p(e)}]) \mapsto^3 ((\sigma(B_1), []), [!_e])$. To prove that \prec -stratification (together with \rightsquigarrow -stratification and \succ -stratification) implies polynomial time, we will need some technical lemmas to handle simplifications of copies.

In the following, we consider a \rightsquigarrow -stratified, \prec -stratified, \succ -stratified proof-net G . Let $s, n \in \mathbb{N}$, we set $T_{s,n} = \{B \in B_G \mid (s_{\rightarrow}(B), s_{\prec}(B)) \leq_{lex} (s, n)\}$ with \leq_{lex} the usual lexicographic order: $(a, b) \leq_{lex} (a', b')$ iff $a < a'$ or $(a \leq a' \text{ and } b \leq b')$. To simplify notations, we write $\mapsto_{s,n}$ for $\mapsto_{T_{s,n}}$, $Cop_{s,n}(C)$ for $Cop_{\mapsto_{T_{s,n}}}(C)$ and so on.

⁹Indeed $S_0 = \emptyset$ and, for every potential edge $(e_l, P_l) \mapsto^0 = (e_l, [e; \dots; e])$. So knowing $(e_l, P_l)^0$ is equivalent to knowing the last edge of the path.

Lemma 40. For $s, n \in \mathbb{N} - \{0\}$ and $(B, P) \in \text{Can}(B_G)$,

$$|\text{Cop}_{s,n}(B, P)| \leq |\text{Can}_{s-1}(\vec{E}_G)|^{\triangleright} \cdot |\vec{E}_G| \cdot \left(\max_{(C, Q) \in \text{Pot}(B_G)} |\text{Cop}_{s,n-1}(C, Q)| \right)^{\partial_G}$$

Proof. If $s \prec(B) > n$, then $((\sigma(B), P), [!_]) \not\mapsto_{s,n}$ and $\text{Cop}_{s,n}(B, P) = \{\mathbf{e}\}$ so the lemma stands. Otherwise (if $s \prec(B) \leq n$), let us consider $t, t' \in \text{Cop}_{s,n}(B, P)$. By definition, there exists paths of the shape:

$$\begin{aligned} ((\sigma(B), P), [!_t]) &\mapsto_{s,n} ((e_1, P_1), T_1) \mapsto_{s,n} \cdots \mapsto_{s,n} ((e_k, P_k), T_k) \mapsto_{s,n} ((e, Q), [!_e]) \\ ((\sigma(B), P), [!_{t'}]) &\mapsto_{s,n} ((e'_1, P'_1), T'_1) \mapsto_{s,n} \cdots \mapsto_{s,n} ((e'_{k'}, P'_{k'}), T'_{k'}) \mapsto_{s,n} ((e', Q'), [!_{e'}]) \end{aligned}$$

By Lemma 38, there are at most $|\text{Can}_{s-1}(\vec{E}_G)|^{\triangleright}$ possible choices for $[e_1; \cdots; e_k]$. Let us suppose that $[e_1; \cdots; e_k] = [e'_1; \cdots; e'_{k'}]$ and $(e, Q)^{s,n-1} = (e', Q')^{s,n-1}$. We will prove by contradiction that $t = t'$.

Let us suppose that $t \neq t'$ and consider them as trees. Because $[e_1; \cdots; e_k] = [e'_1; \cdots; e'_{k'}]$, their leftmost branches are the same. We consider the leftmost branches, b and b' , which are different in t and t' . Let us consider the simplifications u and u' of t and t' whose leftmost branches are b and b' . Thus $u \sqsubset t$, $u' \sqsubset t'$, and the leftmost branches of u and u' are different.

By definition of copies $((\sigma(B), P), [!_u]) \mapsto^* ((_, _), [!_e])$ and $((\sigma(B), P), [!_{u'}]) \mapsto^* ((_, _), [!_{e'}])$. We consider the first step of those paths which differs from the paths corresponding to t and t' . Formally, we consider the lowest $i \in \mathbb{N}$ such that $((\sigma(B), P), [!_u]) \mapsto^i ((f_i, _), _)$ with $f_i \neq e_i$. We are in the following case (with $v \sqsubseteq w$ and $v' \sqsubseteq w'$):

$$\begin{aligned} ((\sigma(B), P), [!_t]) &\mapsto^{i-1} ((\overline{\sigma_a(C)}, P_i), V \cdot !_v) \rightsquigarrow ((e_i, P_i \cdot v), V) \\ ((\sigma(B), P), [!_{t'}]) &\mapsto^{i-1} ((\overline{\sigma_a(C)}, P'_i), V' \cdot !_v) \rightsquigarrow ((e_i, P'_i \cdot v'), V') \\ ((\sigma(B), P), [!_u]) &\mapsto^{i-1} ((\overline{\sigma_a(C)}, P_i), [!_w]) \hookrightarrow ((\sigma(C), P_i), [!_w]) \\ ((\sigma(B), P), [!_{u'}]) &\mapsto^{i-1} ((\overline{\sigma_a(C)}, P'_i), [!_{w'}]) \hookrightarrow ((\sigma(C), P'_i), [!_{w'}]) \end{aligned}$$

We supposed $(e, Q)^{s,n-1} = (e', Q')^{s,n-1}$. By induction on $k-j$, for $1 \leq j \leq k$, we have $((e_j, P_j), T_j)^{s,n-1} = ((e_j, P'_j), T'_j)^{s,n-1}$: we use Theorem 28 for \rightsquigarrow steps and Lemma 36 for \hookrightarrow steps (because $[e_1; \cdots; e_k] = [e'_1; \cdots; e'_{k'}]$). In particular, $((e_i, P_i \cdot v), V)^{s,n-1} = ((e_i, P'_i \cdot v'), V')^{s,n-1}$ so the signatures $((\sigma(C), P_i)^{s,n-1}, [!_v])^{s,n-1}$ and $((\sigma(C), P'_i)^{s,n-1}, [!_{v'}])^{s,n-1}$ are equal.

u is a strict simplification of t so u is not standard. By definition of \prec , we have $B \prec C$ so $s \prec(C) < s \prec(B) \leq n$. One can verify that, for every box D such that $((\sigma(C), Q_i), [!_w]) \mapsto^* ((\sigma(D), _), [!_])$ or $((\sigma(C), Q'_i), [!_{w'}]) \mapsto^* ((\sigma(D), _), [!_])$ we also have $B \prec D$ (so $s \prec(D) \leq n-1$). Thus

$v = ((\sigma(C), P_i), [!_v])^{s,n}$	Because $t \in \text{Cop}_{s,n}(B, P)$
$= ((\sigma(C), P_i), [!_v])^{s,n-1}$	Because “ $s \prec(D) \leq n-1$ ”
$= ((\sigma(C), P'_i), [!_{v'}])^{s,n-1}$	Proved in the previous paragraph.
$= ((\sigma(C), P'_i), [!_{v'}])^{s,n}$	Because “ $s \prec(D) \leq n-1$ ”
$v = v'$	Because $t' \in \text{Cop}_{s,n}(B, P)$

Because $u = u'$ and the $i - 1$ first steps from $((\sigma(B), P), [!_u])$ and $((\sigma(B), P'), [!_{u'}])$ take the same edges (e_1, \dots, e_{i-1}) the leftmost branches of u and u' are equal. This is a contradiction. Our supposition was false, under our assumptions t and t' are equal.

So we proved that, if we choose $[e_1; \dots; e_k]$ and $(e, Q)^{s,n-1}$ then t is uniquely determined. Thus,

$$\begin{aligned} |Cop_{s,n}(B, P)| &\leq |Can_{s-1}(\vec{E}_G)|^{\succ} \cdot |Can_{s,n-1}(\vec{E}_G)| \\ |Cop_{s,n}(B, P)| &\leq |Can_{s-1}(\vec{E}_G)|^{\succ} \cdot |\vec{E}_G| \cdot \left(\max_{(C, Q) \in Pot(B_G)} |Cop_{s,n-1}(C, Q)| \right)^{\partial_G} \end{aligned}$$

□

Theorem 41. Let $x = |\vec{E}_G|$, $S = |\rightsquigarrow|$, $D = |\succ|$, $N = |\swarrow|$, and $\partial = 1 + \partial_G$, then:

$$\max_{(B, P) \in Pot(B_G)} |Cop(B, P)| \leq x^{D^S \cdot \partial^{N \cdot S + N + S} - 1}$$

Proof. For $s, n \in \mathbb{N}$, we set $u_{0,n} = u_{s,0} = 1$ and $u_{s,n} = u_{s-1,N}^{\partial_G \cdot D} \cdot x \cdot u_{s,n-1}^{\partial_G}$. Then, thanks to Lemma 40, we can prove by induction on (s, n) that $\max_{(B, P) \in Pot(B_G)} |Cop_{s,n}(B, P)| \leq u_{s,n}$. One can verify by induction on n that:

$$u_{s,n} = \left(u_{s-1,N}^{\partial_G \cdot D} \cdot x \right)^{\sum_{i=0}^{n-1} \partial_G^i}$$

Thus, for every $n \in \mathbb{N}$,

$$u_{s,n} \leq \left(u_{s-1,N}^{(\partial-1) \cdot D} \cdot x \right)^{\partial^n} = x^{\partial^n} \cdot u_{s-1,N}^{D \cdot \partial^n \cdot (\partial-1)}$$

Thus, we prove by induction on s , that:

$$u_{s,N} \leq \left(x^{\partial^N} \right)^{\sum_{j=0}^{s-1} (D \cdot \partial^N \cdot (\partial-1))^j} \leq \left(x^{\partial^N} \right)^{(D \cdot \partial^N (\partial-1))^s} \leq x^{D^S \cdot \partial^{N \cdot S + N + S} - 1}$$

Finally, let us notice that $\mapsto_{S,N} = \mapsto$, so $Cop_{S,N}(B, P) = Cop(B, P)$. Thus,

$$\begin{aligned} \max_{(B, P) \in Pot(B_G)} |Cop(B, P)| &= \max_{(B, P) \in Pot(B_G)} |Cop_{S,N}(B, P)| \\ &\leq u_{S,N} \\ \max_{(B, P) \in Pot(B_G)} |Cop(B, P)| &\leq x^{D^S \cdot \partial^{N \cdot S + N + S} - 1} \end{aligned}$$

□

Corollary 42. Let us consider a \rightsquigarrow -stratified, \swarrow -stratified, \succ -stratified proof-net G . Let $x = |\vec{E}_G|$, $S = |\rightsquigarrow|$, $D = |\succ|$, $N = |\swarrow|$, and $\partial = 1 + \partial_G$, then:

$$W_G \leq x^{D^S \cdot \partial^{(N+1) \cdot (S+1)}}$$

Proof. By Theorem 41, we have

$$\begin{aligned} W_G &= \left| \text{Can}(\vec{E}_G) \right| \leq \left| \vec{E}_G \right| \cdot \max_{(B,P) \in \text{Pot}(B_G)} |\text{Cop}(B, P)|^\partial \leq x \cdot \left(x^{D^S \cdot \partial^{N \cdot S + N + S - 1}} \right)^\partial \\ W_G &\leq x^{D^S \cdot \partial^{N \cdot S + N + S + 1}} = x^{D^S \cdot \partial^{(N+1)(S+1)}} \end{aligned}$$

□

The polynomial in the bound only depends on $|\rightsquigarrow|$, $|\succ|$, $|\searrow|$, and ∂_G . Those four parameters are bounded by the number of boxes. So a stratified nested proof-net controlling dependence normalizes in a time bounded by a polynomial on the size of the proof-net, the polynomial depending only on the number of boxes of the proof-net.

In the usual encoding of binary words (or other inductive types) in linear logic, the number of boxes is independent of a term. Let us suppose that for every binary word w , the proof-net $(G)_w$ (representing the application of G to the encoding of w) is \rightsquigarrow -stratified, \succ -stratified and \searrow -stratified. Then $W_{(G)_w}$ is bounded by $P(|w|)$ with P a polynomial which does not depend on w . This is the definition of polynomial soundness. However, those semantic criteria are not useful per se:

- The only method we know to check the acyclicity of those relations on a proof-net H is to normalize H to compute the \mapsto -paths. Normalizing a proof-net to obtain a bound on the length of its normalization has no practical use.
- Given a proof-net G , we have no method to check if there exists a binary word such that one of those relation is cyclic on $(G)_w$.

In the next section we will define a decidable subsystem of linear logic (named *SDNLL*) such that \rightsquigarrow , \succ and \searrow are acyclic on every proof-net of *SDNLL*. For $s, d, n \in \mathbb{N}$, we define a formula $\underline{B}_{s,d,n}$ such that every binary word can be encoded by a proof-net typed by $\underline{B}_{s,d,n}$. Determining if a given proof-net G can be typed by a formula of the shape $\underline{B}_{s,d,n} \multimap A$ is decidable. And, if this is the case $(G)_w$ is \rightsquigarrow -stratified, \succ -stratified and \searrow -stratified for every binary word w .

5. Linear logic subsystems and λ -calculus type-systems

5.1. Definition of *SDNLL*

We define a linear logic subsystem, called *SDNLL* (for Stratification Dependence control Nesting Linear Logic) characterizing polynomial time. In *SDNLL*, to enforce \rightsquigarrow -stratification, \succ -stratification and \searrow -stratification, we label the $!$ and $?$ modalities with integers s, d and n . Let us consider a *SDNLL* proof-net G and boxes B and B' with $\beta(\sigma(B)) = !_{s,n,d}A$ and $\beta(\sigma_G(B')) = !_{s',d',n'}A'$. Then we will have the following implications: $(B \rightsquigarrow B' \text{ implies } s > s')$, $(B \succ B' \text{ implies } d > d')$ and $(B \searrow B' \text{ implies } n > n')$. This implies that G is \rightsquigarrow -stratified, \succ -stratified and \searrow -stratified.

Definition 43. For $s \in \mathbb{N}$, we define \mathcal{F}_s by the following grammar (with $t, d, n \in \mathbb{N}$, $t \geq s$ and X ranges over a countable set of variables). Notice that $\mathcal{F}_0 \supseteq \mathcal{F}_1 \supseteq \dots$

$$\mathcal{F}_s := X_t \mid X_t^\perp \mid \mathcal{F}_s \otimes \mathcal{F}_s \mid \mathcal{F}_s \wp \mathcal{F}_s \mid \forall X_t. \mathcal{F}_s \mid \exists X_t. \mathcal{F}_s \mid !_{t,d,n} \mathcal{F}_{t+1} \mid ?_{t,d,n} \mathcal{F}_{t+1}$$

In this section, a *formula context* is a formula where a subterm has been replaced by \circ (e.g. $!_{2,1,3}X \wp \circ$). If h is a formula context and A is a formula, then $h[A]$ refers to the formula obtained by replacing \circ by A in h . We gave another definition for “context” in Section 2.2, and we will define yet another in Section 5.4. Because those terms are well-established terms, we chose not to create new terms. Because these definitions are very different, there should be little confusion.

For any formula of the shape $A = !_{s',d',n'}A'$, we write s_A for s' , d_A for d' and n_A for n' . For $A \in \mathcal{F}_0$, s_A^{\min} refers to the minimum $s \in \mathbb{N}$ such that $A = h[!_{s,_,_}]$ with h a formula context. To gain expressivity, we define a subtyping relation \leq on \mathcal{F}_0 . The relation \leq , defined as the transitive closure of the following \leq^1 relation, follows the intuition that a connective $!_{s,d,n}$ in a formula means that this connective “comes” from a box B with $s_{\rightarrow}(B) \geq s$, $s_{\leftarrow}(B) \geq d$ and $s_{\rightarrow\leftarrow}(B) \geq n$.

$$A \leq^1 B \Leftrightarrow \begin{cases} \text{Either} & A = g[!_{s,d,n}D], B = g[!_{s',d',n'}D], s \geq s', d \geq d' \text{ and } n \geq n' \\ \text{Or} & A = g[?_{s,d,n}D], B = g[?_{s',d',n'}D], s \leq s', d \leq d' \text{ and } n \leq n' \end{cases}$$

Lemma 44. *If $A \leq B$ then $A^\perp \geq B^\perp$*

Proof. Immediate from the definition of \leq . □

Definition 45. A *SDNLL proof-net* is a proof-net whose edges are labelled by a \mathcal{F}_0 formula, the labels respecting the constraints of Figure 15 modulo subtyping.

More precisely, the labelling of a proof-net G is correct if for every node/box whose premises are labelled by A_1, \dots, A_k and whose conclusions are labelled by C_1, \dots, C_l , there exists an instance of the constraint of Figure 15 whose premises are labelled by A_1, \dots, A_k and conclusions are labelled by B_1, \dots, B_l with $B_1 \leq C_1, \dots, B_l \leq C_l$.

For instance, let us suppose that $d_1 \geq d$ and $d_2, \dots, d_k \geq d + 1$ then $?_{s_1,d,n}A_1 \leq ?_{s_1,d_1,n}A_1, ?_{s_2,d+1,n}A_2 \leq ?_{s_2,d_2,n}A_2, \dots, ?_{s_k,d+1,n}A_k \leq ?_{s_k,d_k,n}A_k$. So, a box with premises A_1, \dots, A_k, C and conclusions $?_{s_1,d_1,n}A_1, \dots, ?_{s_k,d_k,n}A_k, !_{s,d,n}C$ satisfies the conditions of *SDNLL* proof-nets.

The *SDNLL* labels are compatible with cut-elimination as shown by the rules of Figure 14. For most rule, the only difficulty is to handle subtyping. We explain it for the \otimes/\wp case (the *ax*, \forall/\exists , $!P/?D$, $!P/?W$ and $!P/?C$ rules are adapted in the same way): by definition of *SDNLL* proof-nets, $C \geq A \wp B$ so C is of the shape $A_1 \wp B_1$ with $A \leq A_1$ and $B \leq B_1$. So $C^\perp = A_1^\perp \otimes B_1^\perp$ and, by definition of *SDNLL* proof-nets, $A_2^\perp \leq A_1^\perp$ and $B_2^\perp \leq B_1^\perp$. By Lemma 44, $A_1^\perp \leq A^\perp$ and $B_1^\perp \leq B^\perp$. So $A_2^\perp \leq A^\perp$ and $B_2^\perp \leq B^\perp$. So, whatever are the nodes with conclusions A_2^\perp and B_2^\perp , we can replace their conclusions with A^\perp and B^\perp without breaking *SDNLL* constraints. One can observe that we could have labelled the edges with $A_1, B_1, A_1^\perp, B_1^\perp$, or with $A_2, B_2, A_2^\perp, B_2^\perp$ or other formulae between A and A_2 and between B and B_2 . We decide not to choose a canonical reduction: this only influences the indices on exponential connectives, and the conclusions of the proof-net are not concerned.

For the sake of readability, in the reductions of Figure 14 we suppose that subtyping only modifies the outermost exponential connectives (modification of labels on inner connectives are dealt as in the \otimes/\wp case). Every letter in the indices represents a

positive number, writing $d + d_1$ as the second index of an edge allows to represent any number greater than (or equal to) d .

- For the $!P/?P$ rule, we can notice that $d = d' + d'_1$ so $d \geq d'$, and $n = n' + n'_1$ so $n \geq n'$. Thus we have $d + d_1 \geq d'$, $d + d_k + 1 > d'$, $n + n_1 \geq n'$ and $n + n_k \geq n'$. The box in the redut satisfies the constraints of $SDNLL$.
- For the $!P/?N$ rule, according to the definition of $SDNLL$ proof-nets, $d \geq d'$, and $n \geq n'$. So $n + n_1 \geq n'$, $n + n_k \geq n'$ and the outermost box of the redut satisfies the constraints of $SDNLL$. We also have $n + n_1 + 1 \geq n' + 1$ and $n + n_k + 1 \geq n' + 1$, so the innermost box of the redut satisfies the constraints of $SDNLL$.

In order to prove the soundness of $SDNLL$ for polynomial time, we first have to prove a technical lemma. Whenever $((e, P), [!_t]@T) \rightsquigarrow^* ((f, Q), [!_u]@U)$, the formulae of e and f are related. To be more precise, if G does not contain any \exists or \forall node, $\beta(e)_T \leq \beta(f)_U$ with $A|_T$ defined as follows:

Definition 46. Let A be a formula and T a trace, we define $A|_T$ by induction on A as follows: $A|_{[]} = A$, $(A \otimes B)|_{T.\otimes_l} = (A \wp B)|_{T.\wp_l} = A|_T$, $(A \otimes B)|_{T.\otimes_r} = (A \wp B)|_{T.\wp_r} = B|_T$, $(\forall X.A)|_{T.\forall} = (\exists X.A)|_{T.\exists} = (!_{s,d,n}A)|_{T.!} = (?_{s,d,n}A)|_{T.?} = A|_T$.

For instance, if $((e, P), T) \rightsquigarrow ((f, P), T.\wp_r)$ then $\beta(e)$ and $\beta(f)$ are of the shape B and $A' \wp B'$ with $B \leq B'$. Let us notice that $\beta(f)_{T.\wp_r} = (B')|_T \geq (B)|_T = \beta(e)_T$.

However, there is a problem with this definition when we cross a \exists link downwards. For example, let us suppose that $((c, []), [\wp_r; !_e; \otimes_r]) \mapsto ((d, []), [\wp_r; !_e; \otimes_r; \exists])$ with $\beta(c) = ?(X \otimes X^\perp) \otimes !(X^\perp \wp X)$ and $\beta(d) = \exists Y.Y \otimes Y^\perp$. Then $\beta(c)_{[\wp_r; !_e; \otimes_r]}$ is undefined: the trace is not compatible with the syntactic tree of $\beta(d)$. In [26], we define a mapping $\beta(\cdot)$ from contexts to formulae (paying special attention to the substitutions caused by the \forall/\exists cut-elimination) satisfying Lemmas 47 to 49:

- The first idea is to substitute, for some of the $\exists X.$ in $\beta(e)$, the occurrences of X by its formula B : if $\beta(e)_T = \exists X.A$ and $((concl_l, _), [\exists]) \rightsquigarrow^* ((e, P), T.\exists@U)$ with l a \exists node whose associated formula is B , we replace $\exists X.A$ by $A[B/X]$.
- Moreover, if $\beta(e)$ contains a free occurrence of a variable X associated with the \forall node m , and $((concl_m, P), [\forall]) \rightsquigarrow^* ((concl_l, _), [\forall])$ with l a \exists node whose associated formula is B , we replace X by B .

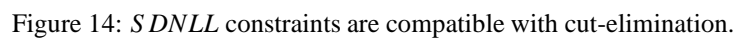
Those two operations can be recursive: the formula B can contain itself free occurrences of variables associated with \forall nodes, or $\exists Y.C$ subformulas. The formal definition can be found in section 5.1.2 of [26].

Lemma 47. If $\beta(e)$ is of the shape $!_{s,d,n}A$, then there exists a substitution θ such that $\beta((e, P), [!_t]@T) = (!_{s,d,n}A[\theta])|_T$.

Lemma 48. If $C \rightsquigarrow^* C'$, we have $\beta(C) \leq \beta(C')$.

The \hookrightarrow step breaks this property: if $\beta(\sigma_i(B)) = ?_{s,d,n}A$ and $\beta(\sigma(B)) = !_{s',d',n'}A'$ then A and A' are a priori unrelated. The only relation required on those formulae is that $d \geq d'$ and $n \geq n'$. Thus, whenever $((\sigma(B), P), [!_t]) \mapsto^* ((\sigma(B'), P'), [!_{t'}])$ with $\beta(\sigma(B)) = !_{s,d,n}A$ and $\beta(\sigma(B')) = !_{s',d',n'}A'$, we have $d \geq d'$ and $n \geq n'$:

Lemma 49. If $C \mapsto^* C'$ and $\beta(C) = !_{s,d,n}A$ then $\beta(C') = !_{s',d',n'}B$ with $d \geq d'$ and $n \geq n'$.



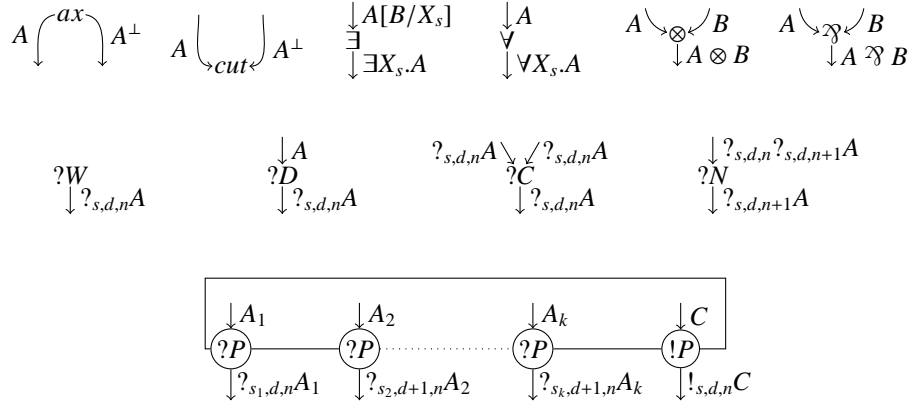


Figure 15: Constraints for $SDNLL$ proof-nets. For the \exists rule, we require $s_B^{min} \geq s$. For the promotion rule, one of the auxiliary door has the same second index as the principal door (in the figure we set arbitrarily this door to be the first one).

5.2. $SDNLL$ is sound for Poly

Thanks to $\beta(-)$, we can prove the implications stated in the beginning of Section 5.1.

Lemma 50. *If $B \rightsquigarrow B'$, $\beta(\sigma(B)) = !_{s,d,n}A$ and $\beta(\sigma(B')) = !_{s',d',n'}A'$ then $s > s'$.*

Proof. Let us suppose that $B \rightsquigarrow B'$, $\beta(\sigma(B)) = !_{s,d,n}A$ and $\beta(\sigma(B')) = !_{s',d',n'}A'$. By definition of \rightsquigarrow , there exist potentials P and P' , and signatures t and u such that $((\sigma(B), P), [!_t]) \rightsquigarrow^* ((\sigma(B'), P'), [!_u]@T) \rightsquigarrow^* ((e, Q), [!_e])$. By Lemma 47, there are substitutions θ and θ' such that $\beta((\sigma(B), P), [!_t]) = !_{s,d,n}A[\theta]$ and $\beta((\sigma(B'), P'), [!_u]@T) = (!_{s',d',n'}A'[\theta'])_T$. So, according to Lemma 48,

$$!_{s,d,n}A[\theta] \leq (!_{s',d',n'}\overline{A'}[\theta'])_T$$

By definition of \leq on formulae, $(!_{s',d',n'}\overline{A'}[\theta'])_T = !_{s'',d'',n''}A''$ with $s \geq s''$. Thus $A'[\theta]$ is of the shape $H[?_{s'',d'',n''}A'']$. Either A' is of the shape $H[?_{s'',d'',n''}A'']$ (in this case by definition of \mathcal{F}_0 , $s' < s''$ so $s > s'$), or there exist sequences $A' = A_0, A_1, \dots, A_k$ of formulae, X^0, \dots, X^k of variables and s_0, \dots, s_k such that for $0 \leq i < k$, A_i is of the shape $H_i[X_{s_i}^i]$ and there exists a \exists node n_i whose associated variable is $X_{s_i}^i$ and whose associated formula is A_{i+1} . And A_k is of the shape $H_k[!_{s'',d'',n''}A'']$. In this case, $s' < s_0 \leq s_1 \leq \dots \leq s_{k-1} \leq s''$ so $s' < s'' \leq s$. \square

Lemma 51. *If $B \succ B'$, $\beta(\sigma(B)) = !_{s,d,n}A$ and $\beta(\sigma(B')) = !_{s',d',n'}A'$ then $d > d'$*

Proof. By definition of \succ , there exists $i \neq j$ and paths of the shape $((\sigma(B), P), [!_t]) \mapsto^+ ((\sigma_i(B'), P'_1), [!_e])$ and $((\sigma(B), P), [!_t]) \mapsto^+ ((\sigma_j(B'), P'_2), [!_e])$. Either $i \neq 1$ or $j \neq 1$. We suppose without loss of generality that $i \neq 1$. By definition of $SDNLL$, $\beta(\sigma_i(B')) = !_{s'',d'',n''}$ with $d'' > d'$. Then, by Lemma 49, $d \geq d'' > d'$. \square

Lemma 52. *If $B \prec B'$, $\beta(\sigma(B)) = !_{s,d,n}A$ and $\beta(\sigma(B')) = !_{s',d',n'}A'$ then $n > n'$*

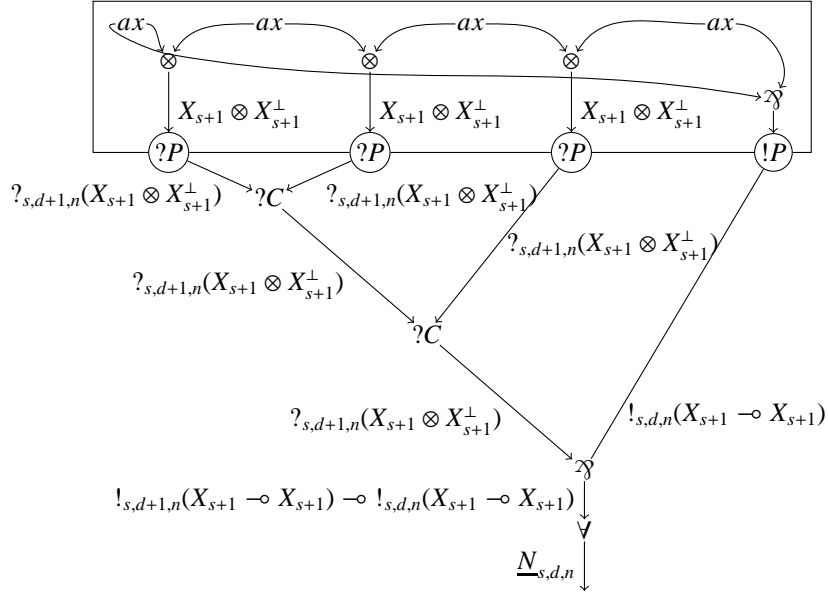


Figure 16: Encoding $\underline{3}_{s,d,n}$ of 3

Proof. By definition of \multimap , there exist $P, Q \in Pot$, and a non-standard signature t , such that $((\sigma(B), P), [!_t]) \mapsto^* ((\sigma(C), Q), [!_e])$. Let us consider the first context of the path such that the leftmost trace element is $!_u$ with u a standard signature. This step must be of the shape $((e, R), [!_{p(u)}]) \rightsquigarrow ((f, R), [!_u])$ with \bar{e} the conclusion of a $?N$ node. By definition of $SDNLL$, $\beta(e) = ?_{\neg n_e}$ and $\beta(f) = ?_{\neg n_f}$ with $n_e > n_f$. Then, by Lemmas 47 and 49, $n \geq n_e > n_f \geq n'$. \square

Corollary 53. *Let G be a $SDNLL$ proof-net, then G is \rightsquigarrow -stratified, \succ -stratified and \multimap -stratified. Moreover, for every $B \in B_G$ with $\beta(\sigma(B)) = !_{s,d,n}A$, $s_{\multimap}(B) \leq s$, $s_{\succ}(B) \leq d$ and $s_{\multimap}(B) \leq n$.*

Proof. Immediate consequence of the three previous lemmas. \square

Theorem 54. *Let G be a $SDNLL$ proof-net, then the maximal reduction length of G (with $x = |\vec{E}_G|$, $\partial = \partial_G$, $S = 1 + \max_{B \in B_G} s_{\sigma(B)}$, $D = \max_{B \in B_G} d_{\sigma(B)}$ and $N = 1 + \max_{B \in B_G} n_{\sigma(B)}$) is bounded by*

$$W_G \leq x^{1+D^S} \cdot \partial^{N \cdot S}$$

Proof. The bound is an immediate consequence of Corollaries 53 and 42. \square

To formalize the polynomial time soundness of $SDNLL$, we need to define an encoding of binary lists. For any $s, d, n \in \mathbb{N}$, we define the formulae $\underline{N}_{s,d,n}$ and $\underline{B}_{s,d,n}$ by

$$\underline{N}_{s,d,n} = \forall X_{s+1}, !_{s,d+1,n}(X_{s+1} \multimap X_{s+1}) \multimap !_{s,d,n}(X_{s+1} \multimap X_{s+1})$$

$$\underline{B}_{s,d,n} = \forall X_{s+1}, !_{s,d+1,n}(X_{s+1} \multimap X_{s+1}) \multimap !_{s,d+1,n}(X_{s+1} \multimap X_{s+1}) \multimap !_{s,d,n}(X_{s+1} \multimap X_{s+1})$$

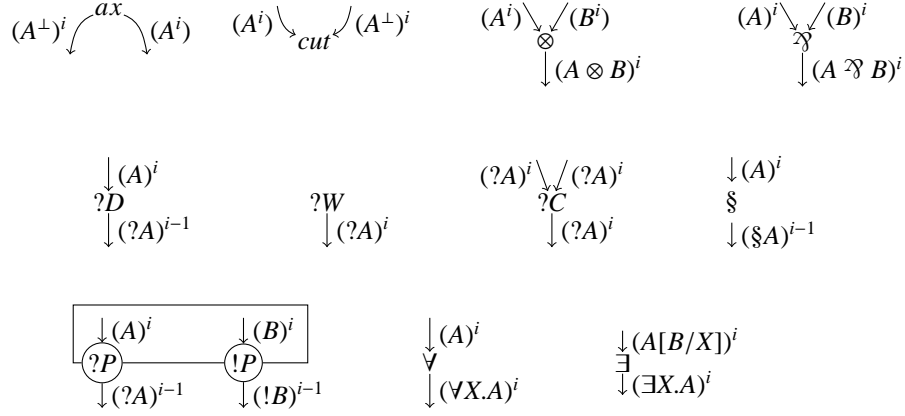


Figure 17: Relations between levels of neighbour edges in L^4 . We also allow boxes with 0 auxiliary doors.

For any $s, d, n \in \mathbb{N}$, $k \in \mathbb{N}$ and binary list l , we can define an encoding $\underline{k}_{s,d,n}$ of k as in Figure 16. The encoding $\underline{l}_{s,d,n}$ of l can be defined similarly. We can verify that the sizes of $\underline{k}_{s,d,n}$ and $\underline{l}_{s,d,n}$ depend linearly on the size of k and l . Finally, for every k and l there is exactly one box in $\underline{k}_{s,d,n}$ and $\underline{l}_{s,d,n}$.

Theorem 55. *For every SDNLL proof-net G whose only conclusion is labelled by $\underline{B}_{s,d,n} \multimap A$, there exists a polynomial P such that for every binary list l ,*

$$W_{(G)\underline{l}_{s,d,n}} \leq P(|l|)$$

Proof. By Theorem 54, $(G)\underline{l}_{s,d,n}$ is \rightsquigarrow -stratified, \succ -stratified and \prec -stratified. The depths of those relations are bounded by $|B_{(G)\underline{l}_{s,d,n}}| = |B_G| + 1$. We can conclude by Corollary 42 (and the linearity of $|E_{\underline{l}_{s,d,n}}|$ on $|l|$). \square

5.3. Encoding of mL^4

There are already many subsystems of LL characterizing polynomial time. We argue that the interest of $SDNLL$ over the previous systems is its intentional expressivity. To support our claim we define an encoding of mL^4 [3]. The encoding of a maximal subsystem of MS [28] is defined in [26]. Baillot and Mazza already proved that LLL can be embedded in mL^4 , thus $SDNLL$ is at least as expressive as the union of those systems.

The formulae of mL^4 are defined as the formulae of linear logic with an additional modality \S and an element of \mathbb{N} indexing the formula. More formally, the set \mathcal{F}_{L^4} of formulae of mL^4 is defined by the following grammar.

$$\begin{aligned} \mathcal{F}_{L^4} &= \mathcal{G}_{L^4} \times \mathbb{N} \\ \mathcal{G}_{L^4} &= X \mid X^\perp \mid \mathcal{G}_{L^4} \otimes \mathcal{G}_{L^4} \mid \mathcal{G}_{L^4} \wp \mathcal{G}_{L^4} \mid \forall X. \mathcal{G}_{L^4} \mid \exists X. \mathcal{G}_{L^4} \mid !\mathcal{G}_{L^4} \mid ?\mathcal{G}_{L^4} \mid \S \mathcal{G}_{L^4} \end{aligned}$$

$$\begin{array}{c}
\frac{}{x : A^\varnothing \vdash x : A} ax \quad \frac{\Gamma \vdash t : A \quad X_s \text{ not free in } \Gamma}{\Gamma \vdash t : \forall X_s. A} \forall_i \quad \frac{\Gamma \vdash t : \forall X_s. A \quad s_B^{min} \geq s}{\Gamma \vdash t : A[B/X]} \forall_e \\
\\
\frac{\Gamma, x : A^\varnothing \vdash t : B}{\Gamma, x : A^{s,d,n} \vdash t : B} ?D \quad \frac{\Gamma \vdash t : B}{\Gamma, x : A^{s,d,n} \vdash t : B} ?W \quad \frac{\Gamma, y : A^{s,d,n}, z : A^{s,d,n} \vdash t : B}{\Gamma, x : A^{s,d,n} \vdash t[x/y; x/z] : B} ?C \\
\\
\frac{\Gamma, x : A^\varnothing \vdash t : B}{\Gamma \vdash \lambda x. t : A \multimap B} \multimap_i \quad \frac{\Gamma, x : A^{s,d,n} \vdash t : B}{\Gamma \vdash \lambda x. t : !_{s,d,n} A \multimap B} \Rightarrow_i \quad \frac{\Gamma \vdash t : A \multimap B \quad \Delta \vdash u : A}{\Gamma, \Delta \vdash (t)u : B} \multimap_e \\
\\
\frac{\Gamma \vdash t : !_{s,d,n} A \multimap B \quad \Delta, \Sigma^\varnothing \vdash u : A \quad d(\Delta \cup \Sigma) \geq d \quad n(\Sigma) \geq n \quad n(\Delta) > n}{\Gamma, \Delta, \Sigma \vdash (t)u : B} \Rightarrow_e
\end{array}$$

Figure 18: $SDNLL_\lambda$ as a λ -calculus type-system.

The index in \mathbb{N} (called *level*) is usually written as an exponent. Intuitively, if the principal edge of B is labelled with $(!A)^s$, the label s represents the stratum of B for \rightsquigarrow . More precisely, it corresponds to a formula of the shape $!_{s+\partial(B), -} A$ in $SDNLL$. Let us notice that, to connect two boxes B and B' labelled with $(!A)^s$ and $(!A')^{s'}$ with $s \neq s'$, we need to use \S nodes.

Let us notice that every box of mL^4 proof-nets have only one auxiliary door. Thus $\succ = \varnothing$ and, for every box B $s_{\succ}(B) = 1$. We can also notice that there is no $?N$ node in mL^4 proof-nets, so for every box B , we have $s_{\prec}(B) = 1$.

We define a mapping $\|\cdot\|$ from formulae contexts of \mathcal{G}_{L^4} to \mathbb{N} which will be used to decide the indices of variables and exponential modalities. For every formulae A in \mathcal{G}_{L^4} and formula context H , $\|\circ\| = 0$, $\|C \otimes H\| = \|H \otimes C\| = \|C \wp H\| = \|H \wp C\| = \|H\|$, $\|\forall X. H\| = \|\exists X. H\| = \|H\|$, and $\|!H\| = \|?H\| = \|\S H\| = 1 + \|H\|$.

Any mL^4 proof-net G can be transformed into a $SDNLL$ proof-net G' as follows: for every variable X appearing in the proof-net, we define M_X as the maximum of the set

$$\{s + \|H\| \mid \beta(e) = (H[X])^s \text{ or } \beta(e) = (H[X^\perp])^s\}$$

Then, we replace every occurrence of X by X_{M_X} . If $\beta(e) = (H[!A])^s$ (resp. $(H[?A])^s$), we replace the modality by $!_{s+\|H\|, 1, 0}$ (resp. $?_{s+\|H\|, 1, 0}$). One can easily verify that G is a valid $SDNLL$ proof-net. The \S node becomes trivial (it does not change the sequent).

The most interesting constraint to check is the constraint on doors. Let us suppose that e is the premise of the i -th auxiliary door of a box B and f is its conclusion. If $\beta_G(e) = (H[!A])^s$ then $\beta_G(f) = ?(H[!A])^{s-1}$. We can notice that $\beta_{G'}(e) = H'[!_{s+\|H\|, 1, 0-}]$ and $\beta_{G'}(f) = H'[!_{s-1+\|H\|, 1, 0-}]$. Those labels are the same because $s + \|H\| = (s-1) + (1 + \|H\|) = (s-1) + \|H\|$.

5.4. $SDNLL$ as a type-system for λ -calculus

As noticed by Baillot and Terui [5], translating naively a subsystem of linear logic into a type-system for λ -calculus can result in a type-system which enjoys neither subject reduction nor the complexity bound enforced by the linear logic subsystem. The subsystem we define is heavily inspired by $DLAL$. For instance, the proof of subject reduction follows the proof of subject reduction of $DLAL$ presented in [6].

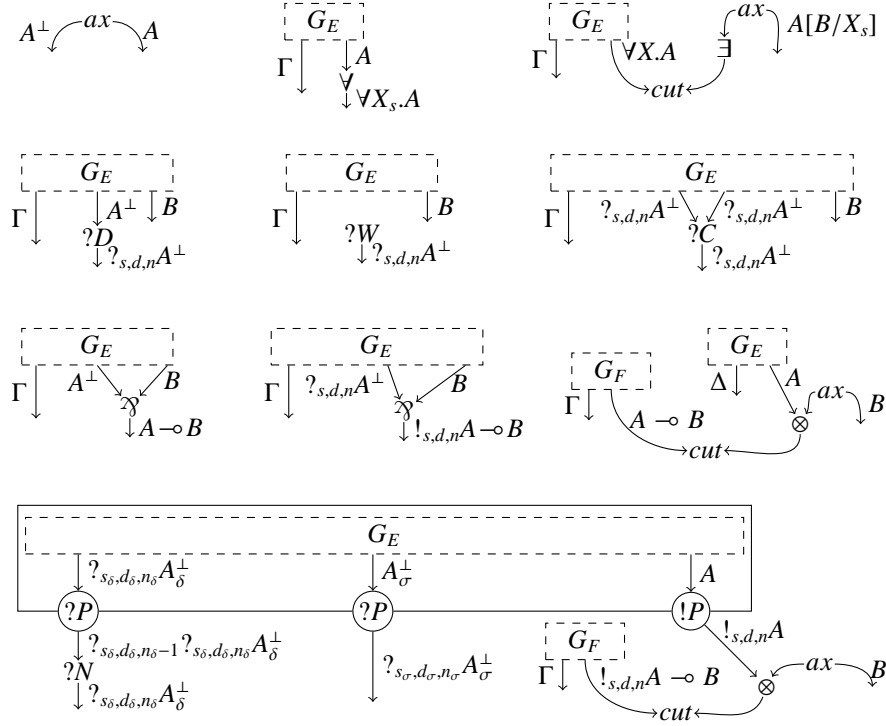


Figure 19: Derivations of $S DNLL_\lambda$ can be translated into $S DNLL$ proof-nets

We restrict the formulae considered by only allowing $!$ modalities on the left side of \multimap connectives.

Definition 56. For $s \in \mathbb{N}$, we define \mathcal{F}_s^λ by the following grammar (with $t, d, n \in \mathbb{N}$, $t \geq s$ and X ranges over a countable set of variables). Notice that $\mathcal{F}_0^\lambda \supseteq \mathcal{F}_1^\lambda \supseteq \dots$

$$\mathcal{F}_s^\lambda := X_t \mid \mathcal{F}_s^\lambda \multimap \mathcal{F}_s^\lambda \mid !_{t,d,n} \mathcal{F}_{t+1}^\lambda \multimap \mathcal{F}_s^\lambda \mid \forall X_t. \mathcal{F}_s^\lambda$$

We define contexts¹⁰ as sets of the shape $\{x_1 : A_1^{l_1}, \dots, x_k : A_k^{l_k}\}$ where the x_i s are pairwise distinct variables of λ -calculus, the A_i s are formulae of \mathcal{F}_0^λ and the l_i s are elements of $\{\emptyset\} \cup \mathbb{N}^3$. Intuitively $A^{s,d,n}$ represents $!_{s,d,n}A$ while A^\emptyset represents A . The set of all contexts is written Con_λ , the set of contexts whose labels are all in \mathbb{N}^3 is written $Con_!$, the set of contexts whose labels are all equal to \emptyset is written Con_\emptyset .

In this paragraph, we consider $\Gamma = \{x_1 : A_1^{s_1, d_1, n_1}, \dots, x_k : A_k^{s_k, d_k, n_k}\} \in Con_!$. Then we write Γ^\emptyset for the context $\{x_1 : A_1^\emptyset, \dots, x_k : A_k^\emptyset\}$. For $s, d, n \in \mathbb{Z}$, we write $\Gamma^{s,d,n}$ for $\{x_1 : A_1^{s_1+s, d_1+d, n_1+n}, \dots, x_k : A_k^{s_k+s, d_k+d, n_k+d}\}$. We write $s(\Gamma)$ for the multiset of left indices, more formally $s(\Gamma) = \{x \mapsto |\{i \in \mathbb{N} \mid s_i = x\}|\}$. We define $d(\Gamma)$ and $n(\Gamma)$ similarly.

¹⁰Because we do not use context semantics in this subsection, there is no ambiguity.

Proof. We prove it by induction on D . The last rule R cannot be in $\{ax, \neg_o, \Rightarrow_e\}$, because the λ -term of the conclusion would be of the shape $\lambda x.t$. If R in $\{\forall_i, \neg_o, \Rightarrow_i\}$, then the lemma is trivial. If R is in $\{?D, ?W, ?C\}$, the derivation is of the shape:

$$\frac{\frac{E}{\Delta \vdash \lambda x.t : A}}{\Gamma \vdash \lambda x.t : A} R$$

By the induction hypothesis, $G_E \rightarrow_{cut}^* G_{E'}$ with E' a derivation of conclusion $\Delta \vdash \lambda x.t : A$ and the last rule of E' introduces the top connective of A . We will examine the case where this last rule is \neg_o , the two other cases are similar. Let us examine the following derivation D'' (let us notice that, because $G_E \rightarrow_{cut}^* G_{E'}$, we have $G_D \rightarrow_{cut}^* G_{D''}$).

$$\frac{\frac{\frac{F'}{\Delta, x : A_1 \vdash t : A_2}}{\Delta \vdash \lambda x.t : A_1 \neg_o A_2} \neg_o}{\Gamma \vdash \lambda x.t : A_1 \neg_o A_2} R$$

In every case we can define D' as the following derivation (let us notice that $G_{D''} = G_{D'}$)

$$\frac{\frac{\frac{F'}{\Delta, x : A_1 \vdash t : A_2}}{\Gamma, x : A_1 \vdash t : A_2} R}{\Gamma \vdash \lambda x.t : A_1 \neg_o A_2} \neg_o$$

The last case to examine is $R = \forall_e$. In this case, the derivation is of the shape:

$$\frac{\frac{E}{\Gamma \vdash \lambda x.t : \forall X_s. A_1}}{\Gamma \vdash \lambda x.t : A_1[B/X_s]} R$$

By the induction hypothesis, $G_E \rightarrow_{cut}^* G_{E'}$ with E' a derivation of conclusion $\Gamma \vdash \lambda x.t : \forall X_s. A$ and the last rule of E' is \forall_i . Let us examine the following derivation D'' (let us notice that, because $G_E \rightarrow_{cut}^* G_{E'}$, we have $G_D \rightarrow_{cut}^* G_{D''}$)

$$\frac{\frac{\frac{F'}{\Gamma \vdash \lambda x.t : A_1}}{\Gamma \vdash \lambda x.t : \forall X_s. A_1} \forall_i}{\Gamma \vdash \lambda x.t : A_1[B/X]} \forall_e$$

Then we can set D' as the $SDNLL$ proof-net obtained from F' by replacing X_s by B in the derivation. We can notice that $D'' \rightarrow_{cut}^2 D'$ (a \forall/\exists step and an axiom step). \square

Lemma 60 (subject reduction). *If there exists a type derivation D whose conclusion is $\Gamma \vdash t : B$ and $t \rightarrow_\beta t'$ then there exists a type derivation D' whose conclusion is $\Gamma \vdash t' : B$ and $G_D \rightarrow_{cut}^+ G_{D'}$.*

Proof. We prove the lemma by induction on D . Because there is a redex in t , t cannot be a variable so the last rule is not an ax rule. Let us suppose that the last rule is a unary rule. Then D is of the shape:

$$\frac{\frac{E}{\Delta \vdash u : A}}{\Gamma \vdash t : B} R$$

$$\begin{array}{c}
\frac{}{g : (X_s \multimap X_s)^\partial \vdash g : X_s \multimap X_s} \quad \frac{}{h : (X_s \multimap X_s)^\partial \vdash h : X_s \multimap X_s} \quad \frac{}{a : (X_s)^\partial \vdash a : X_s} \\
\hline
\frac{}{h : (X_s \multimap X_s)^\partial, a : (X_s)^\partial \vdash (h)a : X_s} \\
\hline
\frac{g : (X_s \multimap X_s)^\partial, h : (X_s \multimap X_s)^\partial, a : (X_s)^\partial \vdash (g)(h)a : X_s}{g : (X_s \multimap X_s)^{s-1,d,n}, h : (X_s \multimap X_s)^\partial, a : (X_s)^\partial \vdash (g)(h)a : X_s} \\
\hline
\frac{g : (X_s \multimap X_s)^{s-1,d,n}, h : (X_s \multimap X_s)^{s-1,d,n}, a : (X_s)^\partial \vdash (g)(h)a : X_s}{f : (X_s \multimap X_s)^{s-1,d,n}, a : (X_s)^\partial \vdash (f)(f)a : X_s} \\
\hline
\frac{f : (X_s \multimap X_s)^{s-1,d,n} \vdash \lambda a.(f)(f)a : X_s \multimap X_s}{\vdash \lambda f.\lambda a.(f)(f)a : !_{{s-1,d,n}}(X_s \multimap X_s) \multimap X_s \multimap X_s} \\
\hline
\vdash \lambda f.\lambda a.(f)(f)a : \forall X_s, !_{{s-1,d,n}}(X_s \multimap X_s) \multimap X_s \multimap X_s
\end{array}$$

Figure 20: Type derivation of $2 : \forall X_s, !_{{s-1,d,n}}(X_s \multimap X_s) \multimap X_s \multimap X_s$.

In every case, u is a subterm of t containing the redex. So, by induction hypothesis, u reduces to a λ -term u' . By the induction hypothesis, there exists a derivation E' of conclusion $\Delta \vdash u' : B$ and $G_E \rightarrow_{cut}^k G_{E'}$ (with $k \geq 1$). We can verify that in every case we can define D' as the following derivation.

$$\frac{\frac{E'}{\Delta \vdash u' : A}}{\Gamma \vdash t' : B} R$$

If the last rule is a \multimap_e or \Rightarrow_e step which does not correspond to the redex, the lemma is proved similarly.

If the last rule is a \multimap_e rule corresponding to the redex then, by Lemma 59, $G_D \mapsto^* G_E$ with E a derivation of the following shape:

$$\frac{\frac{E_l}{\Gamma, x : A \vdash v : B}}{\Gamma \vdash \lambda x.v : A \multimap B} \multimap_i \quad \frac{E_r}{\Delta \vdash u : A}$$

$$\Gamma, \Delta \vdash (\lambda x.v)u : B$$

By Lemma 57, G_E reduces to a derivation of conclusion $\Gamma, \Delta \vdash v[u/x] : B$. If the last rule is a \Rightarrow_e rule corresponding to the redex then, the result follows similarly by Lemmas 59 and 58. \square

Theorem 61. *If there exists a type derivation E whose conclusion is $\Gamma \vdash t : B$, x is the size of E , $S - 1$, $D - 1$ and $N - 1$ are the maximum indexes in E , and ∂ is the depth of E (in terms of \Rightarrow_e rules), then:*

$$t \rightarrow_{\beta}^k t' \quad \Rightarrow \quad k \leq x^{1+D^S} \cdot \partial^{1+N^S}$$

Proof. Immediate from Theorem 54 and Lemma 60. \square

We can notice that, contrary to $DLAL$, $SDNLL_\lambda$ does not allow weakening on linear variables. Thus one can never derive $\vdash \lambda x.t : A \multimap B$ when x is not a free variable of t .

$$\begin{array}{c}
\frac{\frac{m : N \vdash m : N}{m : N \vdash m : !_{s,d,n} F \multimap F} \quad \frac{g : F \vdash g : F}{g : F \vdash g : F} \quad \vdots \quad \frac{x : X_s^\varnothing \vdash x : X_s}{x : X_s^\varnothing \vdash x : X_s}}{m : N, g : F^{s,d,n} \vdash (m)g : F \quad n : N, h : F^{s,d,n}, x : X_s^\varnothing \vdash ((n)h)x : X_s} \\
\frac{m : N, n : N, g : F^{s,d,n}, h : F^{s,d,n}, x : X_s^\varnothing \vdash ((m)g)((n)h)x : X_s}{m : N, n : N, f : F^{s,d,n}, x : X_s^\varnothing \vdash ((m)f)((n)f)x : X_s} \\
\frac{m : N, n : N, f : F^{s,d,n} \vdash \lambda x.((m)f)((n)f)x : X_s \multimap X_s}{m : N, n : N \vdash \lambda f.\lambda x.((m)f)((n)f)x : \underline{N}_{s,d,n}} \\
\frac{m : N, n : N \vdash \lambda f.\lambda x.((m)f)((n)f)x : \underline{N}_{s,d,n} \multimap \underline{N}_{s,d,n}}{\vdash \lambda m.\lambda n.\lambda f.\lambda x.((m)f)((n)f)x : \underline{N}_{s,d,n} \multimap \underline{N}_{s,d,n} \multimap \underline{N}_{s,d,n}}
\end{array}$$

Figure 21: Type derivation of $add : \underline{N}_{s,d,n} \multimap \underline{N}_{s,d,n} \multimap \underline{N}_{s,d,n}$. To simplify the proof derivation, we write F for $X_s \multimap X_s$ and N for $\underline{N}_{s,d,n}$.

We are confident¹¹ that adding the following rule to $SDNLL$ does not break Lemma 61.

$$\frac{\Gamma \vdash t : B}{\Gamma, x : A^\varnothing \vdash t : B}$$

However, one cannot extend the encoding of Figure 19 to this rule because Linear Logic does not allow weakening on a formula A unless A is of the shape $?A'$. Thus we would have to prove the bound directly on λ -calculus (or a similar language as in [5]). Which makes the proof more difficult, because we cannot use the lemmas we proved on context semantics. If we had defined the context semantics and the criteria on a more general framework (for example interaction nets, for which we define a context semantics in [25]) we would not have problems to accomodate such a simple modification.

To give an intuition on the system, let us give some examples of proof derivations. For any $s \geq 1$, and $k \in \mathbb{N}$, \underline{k} can be typed with the following type (see Figure 20 for the type derivation of $\underline{2}$):

$$\mathbf{N}_{s,d,n} = \forall X_s, !_{s-1,d,n}(X_s \multimap X_s) \multimap X_s \multimap X_s$$

Addition can be typed as shown in Figure 21. Finally, although this type system has no built-in mechanism to type tuples, we can encode them by the usual church encoding (Figure 22). Let us notice that this encoding does not require any additional constraint on the types, contrary to mL^4 (where the terms must have the same level).

We isolate four constraints that previous logics (LLL , SLL , and MS) has and which $SDNLL$ does not have. We illustrate each constraint with an intuitive description and a λ -term which can be typed in $SDNLL$ but seemingly not in previous logics because of this constraint. We set $S = \lambda m.\lambda f.\lambda x.((m)f)(f)x$ implementing successor, and $+$ as

¹¹Adding generalized weakening (Conclusion of $?C$ can be any formula) would not change a single line in the proof of Theorem 55. However, in order to prove Lemma 60, we would need to add cut-elimination rules: $?C$ cut with any node n , deletes n and creates $?C$ nodes cut with every premise of n . W_G would still be a valid bound on the number of steps during reduction, so Sections 3 and 4 would be the same.

$$\begin{array}{c}
\frac{f : (A \multimap B \multimap X)^\circ \vdash f : A \multimap B \multimap X \quad \Gamma \vdash t : A}{\Gamma, f : (A \multimap B \multimap X)^\circ \vdash (f)t : B \multimap X} \quad \Delta \vdash u : B \\
\hline
\frac{\Gamma, \Delta, f : (A \multimap B \multimap X)^\circ \vdash ((f)t)u : X}{\Gamma, \Delta \vdash \lambda f.((f)t)u : (A \multimap B \multimap X) \multimap X} \\
\hline
\frac{\Gamma, \Delta \vdash \lambda f.((f)t)u : \forall X.(A \multimap B \multimap X) \multimap X}{\Gamma, \Delta \vdash \langle t, u \rangle : \langle A, B \rangle}
\end{array}$$

Figure 22: Simple encoding of pairs.

the λ -term $\lambda m.\lambda n.\lambda f.\lambda x.((m)f)((n)f)x$ implementing addition on Church integers and $x + y + z$ is a notation for $((+)((+)x)y)z$.

- In previous logics, in $\langle t, u \rangle$, t and u must have the same stratum indices (depth in *LLL* and *MS*, level in mL^4). The term $\underline{k}\lambda\langle x, y, z \rangle.\langle x, ((x)S)\underline{0}, x + y + z \rangle$ is not typable in previous logics: because the function $\lambda\langle x, y, z \rangle.\langle x, ((x)S)\underline{0}, x + y + z \rangle$ is iterated, we have $s(y) = s(((x)S)\underline{0})$ so $s(y) > s(x)$. But, because they are in the same tuple, it must be $s(y) = s(x)$. The $x + y + z$ term ensures that the stratum indices of y and x cannot be modified by § modalities.
- There is no N rule in previous logics and in their encodings in *SDNLL*. This seems to prevent the typing of $\underline{k}(\lambda\langle v, w, x, y, z \rangle.\langle w, w, x, w + x + y, ((x)(+)v)\underline{0} \rangle)$.
- Contrary to *LLL* and mL^4 , one can have several variables in the context during a \Rightarrow_e rule. So, $t = \underline{k}(\lambda\langle x, y, z \rangle.\langle x, x + y, y \rangle)$ is typable in *SDNLL* but not in *LLL* and mL^4 . Moreover, the maximum nest of terms is not a priori bounded by the type system, so if we set $u = \lambda\langle x, y, z \rangle.\langle z, z, z \rangle$, then $(t)(u)(t) \cdots (u)t$ is typable in *SDNLL* whatever the length of the chain of applications, whereas in *MS* the maximum length of such a chain is bounded.
- Previous logics had no subtyping. For example, in mL^4 , a A^i formula cannot be considered as a A^{i-1} formula. The example in the first item of this list would be typable in mL^4 if it was allowed to decrease the level of a formula by mean of a subtyping relation.

6. Conclusion and further work

In order to address the potential applications given in the introduction (real-time systems, complexity debugging, mathematical proofs) we aim to create a type system for a programming language such that:

1. Programming in the language is practical. The language offers usual features such as built-in types (integers, boolean,...), control flow operations, recursive definitions, side effects,...
2. Type inference is decidable in reasonable time.

3. For most polynomial time program users will write, the type inferred entails a polynomial bound.
4. The bounds inferred are often tight (very important for real-time systems, rather important for complexity debugging, unimportant for mathematical proofs).

We consider that goals 2 and 4 highly depend on the system. We could try to design a faster type inference algorithm for *LLL*, or infer tighter bounds for a *LLL* program. However, because of its lack of expressivity, there is little chance that *LLL* will be used in practice for the goals we have in mind. A new system must be created, and type inference and tight bound inference may be totally different in this new system.

We view goals 1 and 3 as mostly orthogonal. It is possible to define an expressive functional core (a linear logic subsystem or λ -calculus type-system) and add features to it. For example, previous works have added pattern-matching, recursive definitions [2], side-effects and concurrent features [21] to *LLL*. However, previous works only extended a specific system (*LLL* in those cases). We are not aware of any work proving that “for every subsystem *S* of linear logic sound for *Ptime* and verifying some condition *C*, the system obtained by adding feature *F* to *S* is sound for *Ptime*”. So, if we added other features to *LLL* without breaking the polynomial bound, it is unclear whether we would have been able to add those features to other subsystems of linear logic characterizing polynomial time. In those conditions, it made more sense to first work on the functional core (goal 3), and in a second step, add features to it (goal 1).

Because *SDNLL* is more expressive than previous subsystems of linear logic characterizing polynomial time, this work fits into goal 3. However, in the same way Baillot and Mazza considered that the “fundamental contribution of” [3] is not the definition of the systems mL^3 and mL^4 themselves, but the demonstration that “in linear-logical characterizations of complexity classes, exponential boxes and stratification levels are two different things”, we consider that the main contribution of our work is the idea to define semantic criteria based on the acyclicity of relation on boxes:

- Using those criteria, we separated three principles underlying *LLL* and the works based on it. It sheds a new light on previous works: mL^4 relaxes the “stratification” criterion of *LLL*, while *MS* relaxes its “dependence control” criterion (we are not aware of previous works relaxing the “nesting” condition). Realizing that those principles are mainly orthogonal can help further works on the expressivity of linear logic subsystems characterizing polynomial time: independent improvements on different principles can be combined. For instance, one can easily verify, that one can combine mL^4 and a maximal *Ptime* system of *MS*¹². In fact, *SDNLL* can be seen as an extension of such a system.
- Because the lemmas and results of sections 2, 3 and 4 are valid for any untyped proof-net, they can be reused to prove polynomial bounds for other subsystems of linear logic in which \rightsquigarrow , \succ or \prec are acyclic (such as mL^4 , *MS* and the multiplicative fragment of *LLL*) or to define new criteria: in [26], Perrinel builds

¹² mL^4 with its “at most one auxiliary door by box” replaced by the indices criteria of the *MS* system to control dependence

upon these technical lemmas to define more expressive criteria entailing a polynomial bound and a criterion entailing a primitive recursive bound.

- We separated the task of creating an expressive subsystem of linear logic characterizing polynomial time into subtasks: finding loose criteria on semantic entailing polynomial time, and finding syntactic criteria entailing those semantic criteria. One can closely examine the proofs leading to Corollary 42, to find any unnecessary assumption on the proof-net behaviour. While this may be subjective, we found it much easier to reason about complex semantic criteria without having to consider the exact way in which they will be enforced.
- While the syntax of *SDNLL* (or any other syntactic subsystem of linear logic) may be difficult to adapt to richer languages where the notion of reduction differs from cut-elimination, those relations on boxes have a meaning going beyond linear logic itself: $B \rightsquigarrow C$ means that B interacts with an element created by an interaction of C (with nodes created when C is opened/interacts), $B \gg C$ and $B \ll C$ represent two ways of having several duplicates of B inside C . Thus it would be interesting to investigate the application of those principles to other models of computation based on reduction/rewriting.

The applications considered in the introduction are used to motivate the direction of our research, to explain why the intensional expressivity of our characterization is an important problem. We are still far from having a system expressive enough to handle them. We explained why we first focused on the expressivity of the functional core (subsystems of linear logic and type systems on plain λ -calculus), but we consider that the main challenge in the future of Implicit Computational Complexity will be to add features to this functional core (built-in types, pattern-matching, recursive definitions, side-effects,...) to get closer to the programming languages used in practice. Thus, the fact that those criteria might be easier to adapt to a more practical framework is especially important.

In a previous work [25], we defined a context semantics for interaction nets: a well-behaved class of graph rewriting systems [19] based on proof-nets. Interaction net is not a single system, but a set of such systems. Thus, this framework seems particularly adapted to the progressive addition of features. An interesting problem for future work would be to use the context semantics of [25] to define relations on interaction nets corresponding to \rightsquigarrow , \gg and \ll .

7. Bibliography

References

- [1] V. Atassi, P. Baillot, and K. Terui. Verification of ptime reducibility for system F terms: Type inference in dual light affine logic. *Logical Methods in Computer Science*, 3(4), 2007.
- [2] P. Baillot, M. Gaboardi, and V. Mogbil. A polytime functional language from light linear logic. *Programming Languages and Systems*, 2010.

- [3] P. Baillot and D. Mazza. Linear logic by levels and bounded time complexity. *Theoretical Computer Science*, 411(2), 2010.
- [4] P. Baillot and M. Pedicini. Elementary complexity and geometry of interaction. *Fundamenta Informaticae*, 45(1-2), 2001.
- [5] P. Baillot and K. Terui. Light types for polynomial time computation in lambda calculus. *Information and Computation*, 207(1), 2009.
- [6] Patrick Baillot and Kazushige Terui. Light types for polynomial time computation in lambda-calculus. *CoRR*, cs.LO/0402059, 2004.
- [7] U. Dal Lago. The geometry of linear higher-order recursion. In *Logic in Computer Science, 2005. LICS 2005. Proceedings. 20th Annual IEEE Symposium on*. IEEE, 2005.
- [8] U. Dal Lago. Context semantics, linear logic, and computational complexity. *ACM Transactions on Computational Logic*, 10(4), 2009.
- [9] U. Dal Lago and M. Hofmann. Bounded linear logic, revisited. In *Typed Lambda Calculi and Applications*. Springer, 2009.
- [10] V. Danos and J.B. Joinet. Linear logic and elementary time. *Information and Computation*, 183(1), 2003.
- [11] V. Danos and L. Regnier. Proof-nets and the Hilbert space. *London Mathematical Society Lecture Note Series*, 1995.
- [12] J.Y. Girard. Une extension de l'interpretation de gödel a l'analyse, et son application a l'elimination des coupures dans l'analyse et la theorie des types. *Studies in Logic and the Foundations of Mathematics*, 63, 1971.
- [13] J.Y. Girard. Linear logic. *Theoretical computer science*, 50(1), 1987.
- [14] J.Y. Girard. Proof-nets: the parallel syntax for proof-theory. *Logic and Algebra*, 180, 1996.
- [15] J.Y. Girard. Light linear logic. *Information and Computation*, 143(2), 1998.
- [16] J.Y. Girard, A. Scedrov, and P.J. Scott. Bounded linear logic: a modular approach to polynomial-time computability. *Theoretical computer science*, 97(1), 1992.
- [17] G. Gonthier, M. Abadi, and J.J. Lévy. The geometry of optimal lambda reduction. In *Proceedings of the 19th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM, 1992.
- [18] G. Gonthier, M. Abadi, and J.J. Lévy. Linear logic without boxes. In *Logic in Computer Science, 1992. LICS'92., Proceedings of the Seventh Annual IEEE Symposium on*. IEEE, 1992.
- [19] Y Lafont. Interaction nets. In *Principles of programming languages, 17th ACM SIGPLAN-SIGACT symposium on*. ACM, 1989.

- [20] Y. Lafont. Soft linear logic and polynomial time. *Theoretical Computer Science*, 318(1-2), 2004.
- [21] Antoine Madet. A polynomial time λ -calculus with multithreading and side effects. In Danny De Schreye, Gerda Janssens, and Andy King, editors, *Principles and Practice of Declarative Programming, PPDP'12, Leuven, Belgium - September 19 - 21, 2012*, pages 55–66. ACM, 2012.
- [22] Antoine Madet and Roberto M Amadio. An elementary affine λ -calculus with multithreading and side effects. In *Typed Lambda Calculi and Applications*. Springer, 2011.
- [23] D. Nowak and Y. Zhang. Formal security proofs with minimal fuss: Implicit computational complexity at work. *Information and Computation*, 2014.
- [24] M. Perrinel. On paths-based criteria for polynomial time complexity in proof-nets. In *FOPARA*, volume 8552 of *Lecture Notes in Computer Science*, pages 127–142. Springer, 2013.
- [25] M. Perrinel. On context semantics and interaction nets. In T.A. Henzinger and D. Miller, editors, *CSL-LICS*, pages 73:1–73:10. ACM, 2014.
- [26] M. Perrinel. *Investigating the expressivity of linear logic subsystems characterizing polynomial time*. PhD thesis, Laboratoire de l’informatique du parallisme, cole Normale Suprieure de Lyon, France, juillet 2015 http://perso.ens-lyon.fr/matthieu.perrinel/pub/these_perrinel.pdf.
- [27] L. Roversi and L. Vercelli. Some complexity and expressiveness results on multimodal and stratified proof nets. In *Types for Proofs and Programs*. Springer, 2009.
- [28] L. Roversi and L. Vercelli. A local criterion for polynomial-time stratified computations. In *Foundational and Practical Aspects of Resource Analysis*. Springer, 2010.
- [29] Y. Zhang. The computational SLR: A logic for reasoning about computational indistinguishability. In *Typed lambda calculi and applications: 9th international conference, TLCA 2009, Brasília, Brazil, July 1-3, 2009: proceedings*, volume 5608. Springer-Verlag New York Inc, 2009.